Google DeepMind

Has Machine Learning for Systems Reached an Inflection Point?

Martin Maas (Google DeepMind)

ASPLOS & EuroSys Plenary Session (April 1, 2025) – Keynote

Presenting the work of <u>many</u> people.

ML Adoption Across Fields

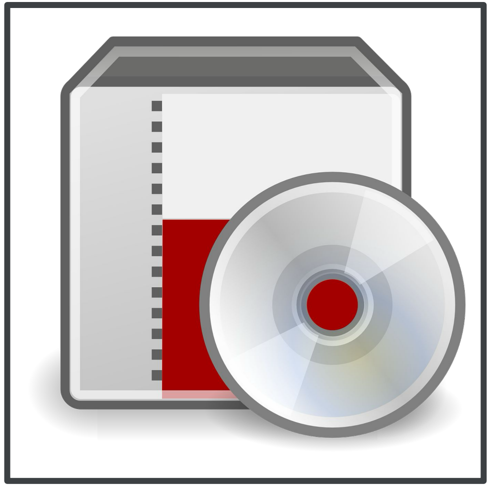
ML has revolutionized a number of different fields.

	1960	2025
Computer Vision	Wavelets, SIFT, Eigenfaces,... + some ML	Mostly ML (neural networks)
	AlexNet (2012)	
NLP	Symbolic/statistical methods + some ML	Mostly ML (neural networks)
	Word2Vec (2013)	
Code (PL, SW Eng.)	Heuristics, solvers, syntax trees + some ML	Increasingly ML/AI
	LLMs (2022)	

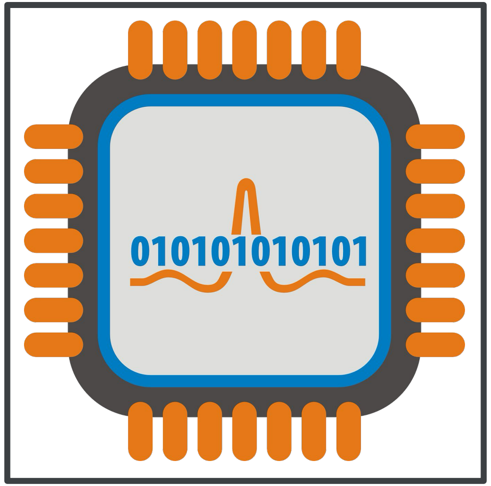
What About Systems?

Learning-Based Systems

Data Center Scheduling



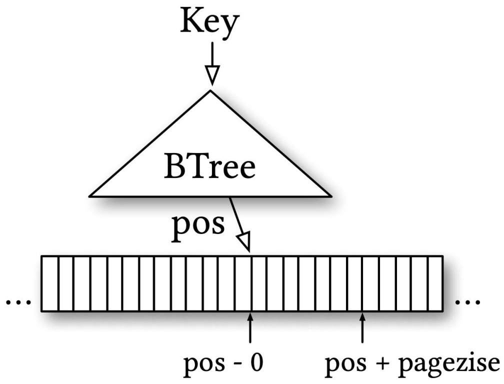
Compilers & Runtimes



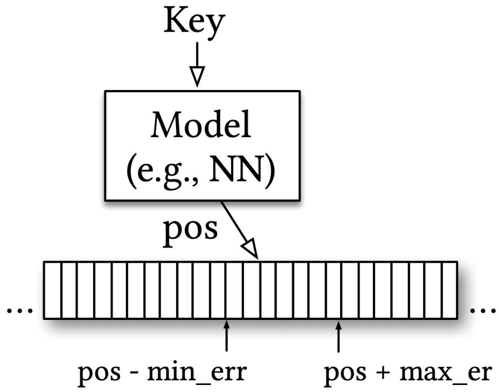
Chip Design

Example: Learned Index

The Case for Learned Index Structures, Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. (SIGMOD '18).



B-Tree Index



Learned Index

ML for Systems Community

ML for Systems has evolved into a community.

- **Technical area** within **ASPLOS**, **EuroSys**, **MLSys**.
- **Workshops:** ML for Systems (NeurIPS), EuroMLSys (EuroSys), PACMI (SOSP) and others.
- **Research Initiatives**: **Architecture 2.0**, Learning Directed Operating System (**LDOS**) and others.

Learning-Based Systems

Learning-based systems are showing **clear promise**.
What will be the **catalyst** driving widespread adoption?

There may not be a single answer.

Talk Outline

1 Conceptual Abstractions

Standardized ways for building learning into systems

2 ML Support in Systems

Best practices for deploying learning-based systems

3 Growing AI Capabilities

GenAI and other approaches

Talk Outline

1 Conceptual Abstractions

Standardized ways for building learning into systems

2 ML Support in Systems

Best practices for deploying learning-based systems

3 Growing AI Capabilities

GenAI and other approaches

Why do abstractions matter?

In other areas, clear abstractions enabled progress and principled approaches:

- Scheduling, Compiler Passes, Memory Allocation,...

In contrast, ML for Systems often requires significant amounts of ad-hoc work.

Analogy: Distributed Systems

Building a distributed system used to be very challenging. Algorithms and protocols had to be custom-built.

Grapevine: An Exercise in Distributed Computing
Andrew D. Birrell, Roy Levin, Roger M. Needham, and Michael D. Schroeder (Xerox PARC)

Consensus as a clear **abstraction** facilitated building of distributed systems. Consensus protocols (e.g., Paxos) and systems built on top of them evolved in parallel.

Today, we can build on standard frameworks and libraries.

Challenge: "ML for Systems" refers to a very wide range of different things.

To define **abstractions for ML for systems**, we need to be clear what ML **is used for**. We need a **taxonomy**.

Definitions

System Policy: Given a software or hardware component that makes decisions related to the execution of computer programs, a system policy describes how these decisions are made.

Learning-Based Systems: Systems that use machine learning in the implementation of a system policy.

Martin Maas, "**A Taxonomy of ML for Systems Problems**," in IEEE Micro, vol. 40, no. 5, pp. 8-16, 1 Sept.-Oct. 2020, DOI: 10.1109/MM.2020.3012883.

Google

Dimension 1: Application Area

- ML for databases: learned index structures, query optimization
- ML for compilers: cost models, vectorization
- ML for hardware design: chip placement, HW/SW co-design
- ML for accelerator design: neural architectures, exploration
- ML for memory management (and garbage collection)
- ML for cluster scheduling, resource allocation
- ML for configuration parameters tuning
- ML for prefetching, branch prediction
- ML for failure detection/prevention, performance regressions
- ML for network routing

Dimension 2: How ML is Used

What does ML enable that a conventional approach could not do? **(Not every problem benefits from ML.)**

Anomaly Detection (e.g., detecting performance regressions)
Forecasting (e.g., predicting future application resource demands)
Extrapolation (e.g., classifying programs as scale-up or scale-out)
Discovery (e.g., coming up with new caching policies)
Optimization (e.g., ML for hardware design, autotuners)

Classifying Existing Work

	Anomaly Detection	Forecasting	Extrapolation	Discovery	Optimization
Compiler Optimization					
Query Optimization					
Hardware Design					
Cluster Scheduling					
Memory Allocation					
Networking					
Prefetching					

Classifying Existing Work

	Anomaly Detection	Forecasting	Extrapolation	Discovery	Optimization
Compiler Optimization					
Query Optimization					
Hardware Design					
Cluster Scheduling					
Memory Allocation					
Networking					
Prefetching					

A learned performance model for tensor processing units, Kaufman et al. (MLSys'21)

Classifying Existing Work

	Anomaly Detection	Forecasting	Extrapolation	Discovery	Optimization
Compiler Optimization					
Query Optimization					
Hardware Design					
Cluster Scheduling					
Memory Allocation					
Networking					
Prefetching					

Seer: Leveraging Big Data to Navigate the Complexity of Performance Debugging in Cloud Microservices
Gan et al. (ASPLOS'19)

Classifying Existing Work

	Anomaly Detection	Forecasting	Extrapolation	Discovery	Optimization
Compiler Optimization					
Query Optimization					
Hardware Design					
Cluster Scheduling					
Memory Allocation					
Networking					
Prefetching					

Learning Memory Access Patterns
Hashemi et al. (ICLR, 2018)

Classifying Existing Work

	Anomaly Detection	Forecasting	Extrapolation	Discovery	Optimization
Compiler Optimization					
Query Optimization					
Hardware Design					
Cluster Scheduling					
Memory Allocation					
Networking					
Prefetching					

Challenge: Quadratic number of areas, each requiring new data sets, libraries and interfaces.

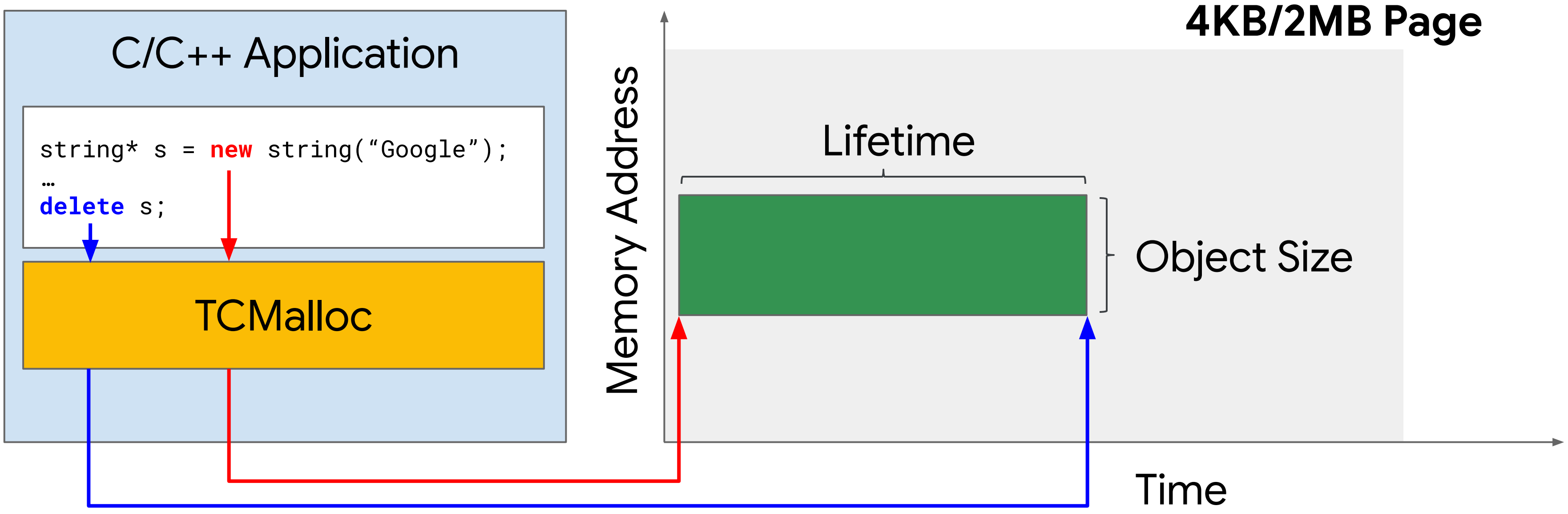
Conceptual Abstractions

Can we create **reusable recipes** that apply across a wide range of different domains?

- How to translate ML predictions into system decisions.
- How to tolerate ML prediction errors.
- How to handle noisy and unpredictable data.
- How to handle workloads that drift over time.
- How to solve NP-complete problems with ML.

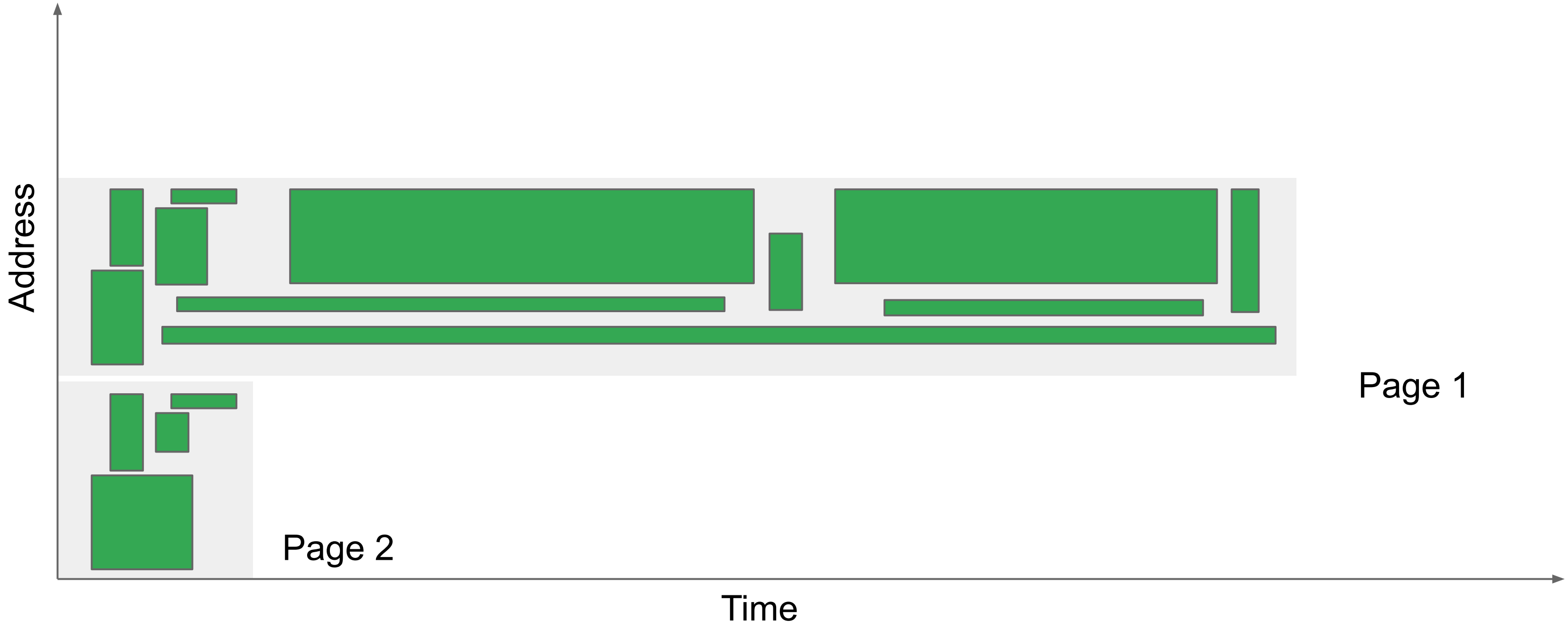
Let’s look at an example.

Example: Memory Allocation



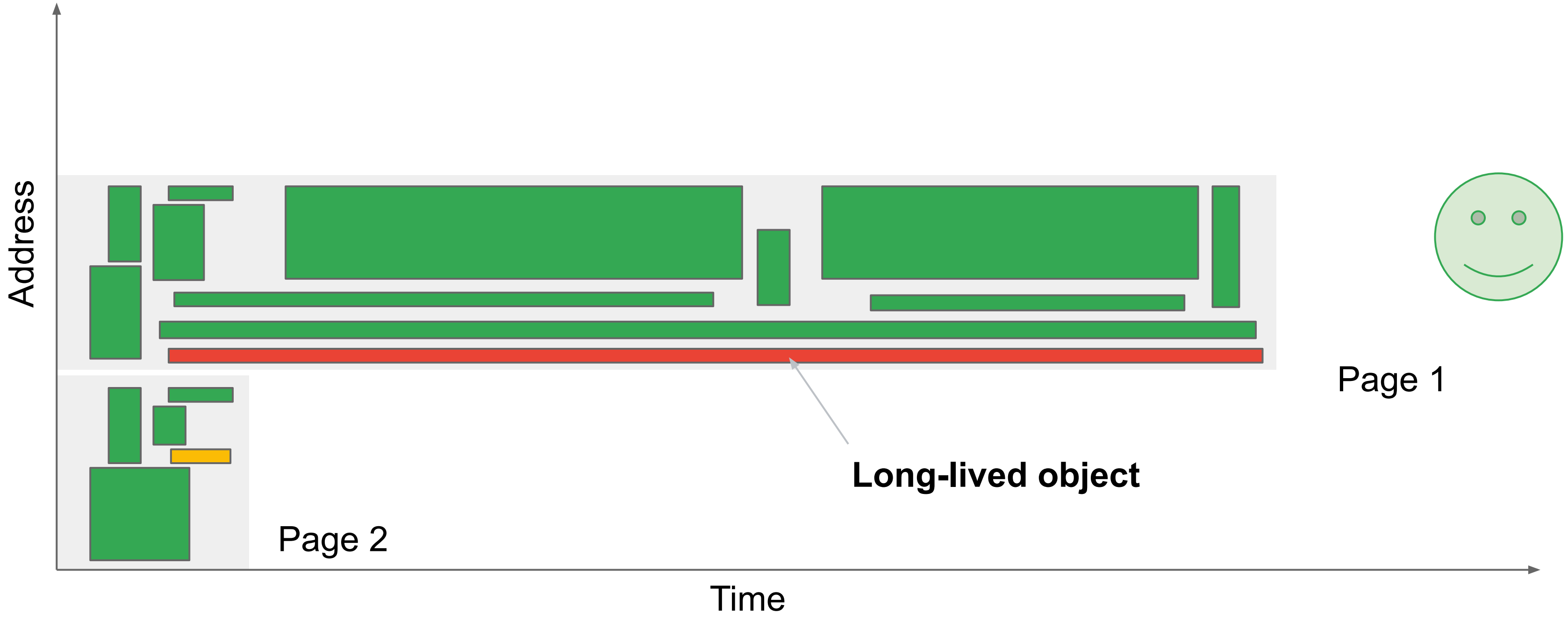
"**Learning-based Memory Allocation for C++ Server Workloads**", ASPLOS 2020, Martin Maas, David G. Andersen, Michael Isard, Mohammad Mahdi Javanmard, Kathryn S. McKinley, Colin Raffel

Huge Page Fragmentation



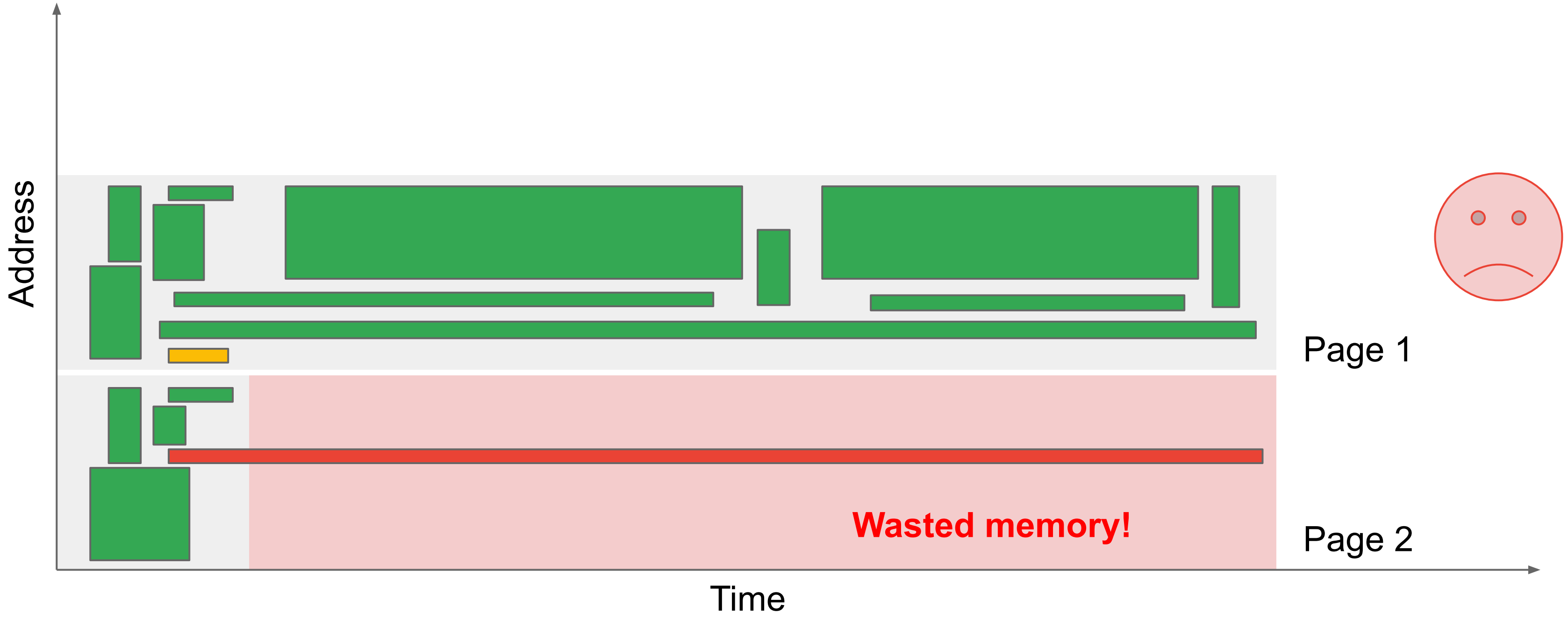
"**Learning-based Memory Allocation for C++ Server Workloads**", ASPLOS 2020, Martin Maas, David G. Andersen, Michael Isard, Mohammad Mahdi Javanmard, Kathryn S. McKinley, Colin Raffel

Huge Page Fragmentation



“**Learning-based Memory Allocation for C++ Server Workloads**”, ASPLOS 2020, Martin Maas, David G. Andersen, Michael Isard, Mohammad Mahdi Javanmard, Kathryn S. McKinley, Colin Raffel

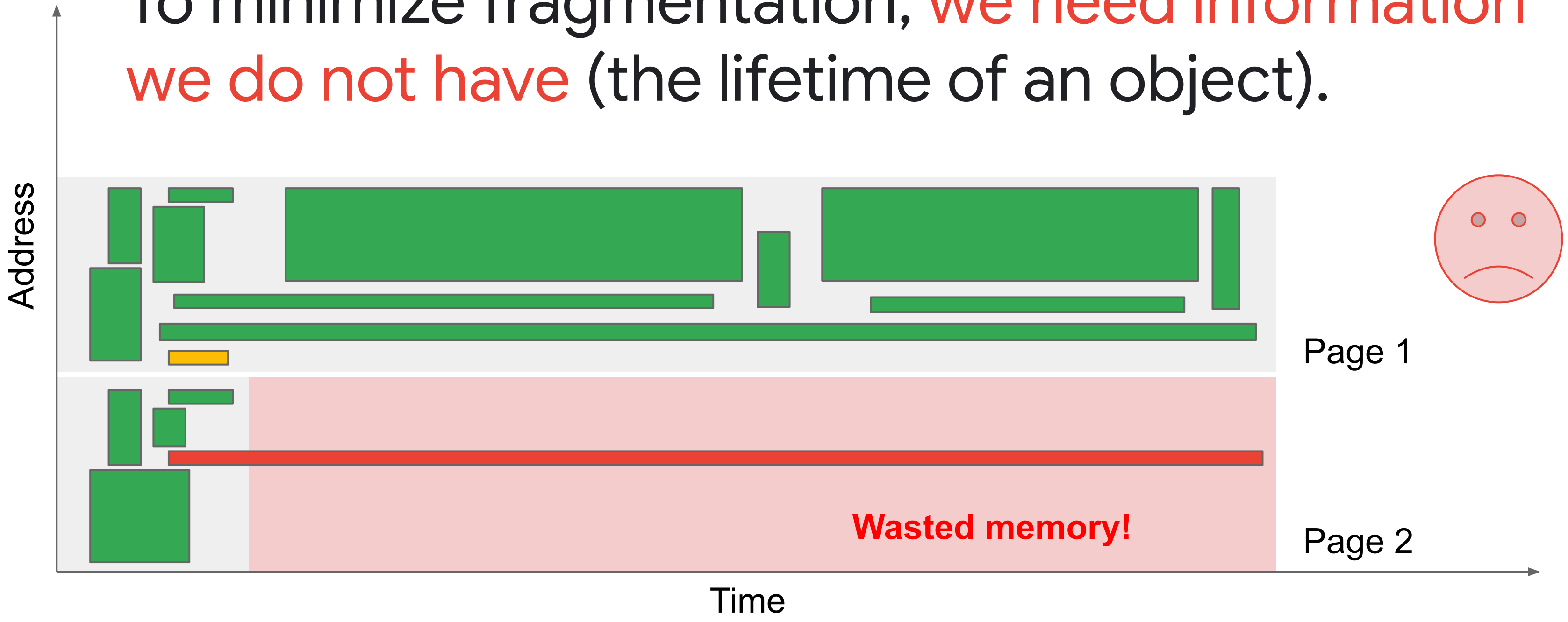
Huge Page Fragmentation



"**Learning-based Memory Allocation for C++ Server Workloads**", ASPLOS 2020, Martin Maas, David G. Andersen, Michael Isard, Mohammad Mahdi Javanmard, Kathryn S. McKinley, Colin Raffel

Huge Page Fragmentation

To minimize fragmentation, we need information we do not have (the lifetime of an object).



"**Learning-based Memory Allocation for C++ Server Workloads**", ASPLOS 2020, Martin Maas, David G. Andersen, Michael Isard, Mohammad Mahdi Javanmard, Kathryn S. McKinley, Colin Raffel

Using ML for Forecasting

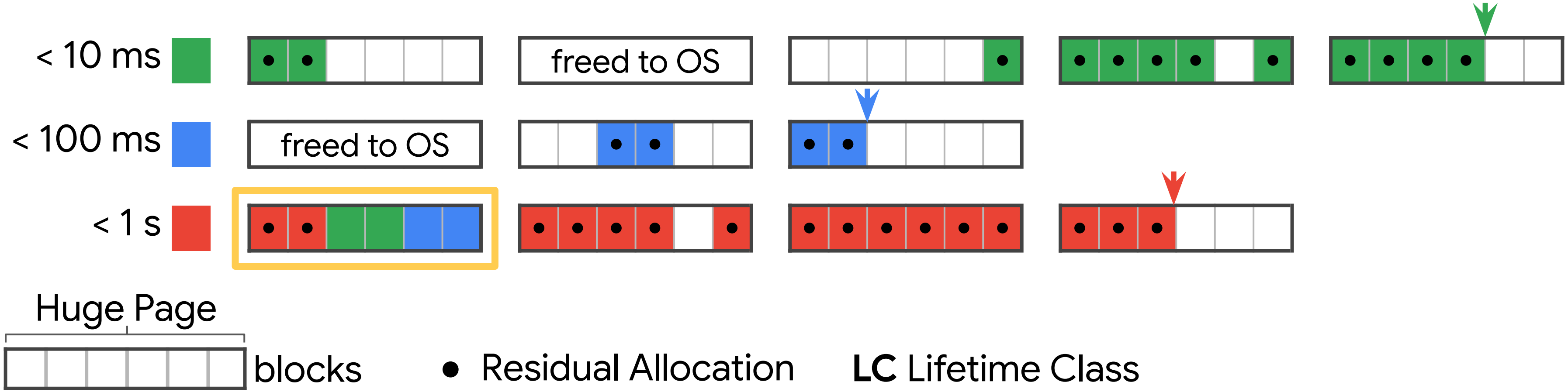
Symbols within stack traces contain meaning and encode programmer intent. Apply ML to this information in order to predict **object lifetimes** that the allocator then uses.

```
 1.  __gnu_cxx::__g::__string_base<char, std::__g::char_traits<char>,
     std::__g::allocator<char> >::_M_reserve(unsigned long)
 2.  proto2::internal::InlineGreedyStringParser(std::__g::basic_string<char,
     std::__g::char_traits<char>, std::__g::allocator<char> >*, char const*,
     proto2::internal::ParseContext*)
 3.  proto2::FileDescriptorProto::_InternalParse(char const*,
     proto2::internal::ParseContext*)
 4.  proto2::MessageLite::ParseFromArray(void const*, int)
 5.  proto2::DescriptorPool::TryFindFileInFallbackDatabase(std::__g::basic_string<char,
     std::__g::char_traits<char>, std::__g::allocator<char> > const&) const
 6.  proto2::DescriptorPool::FindFileByName(std::__g::basic_string<char,
     std::__g::char_traits<char>, std::__g::allocator<char> > const&) const
 7.  proto2::internal::AssignDescriptors(proto2::internal::AssignDescriptorsTable*)
 8.  store2::Algorithm_descriptor()
 9.  store2::init_module_algorithm_parse()
10.  Initializer::TypeData::RunIfNecessary(GoogleInitializer*)
11.  Initializer::RunInitializers(char const*)
12.  RealInit(char const*, int*, char***, bool, bool)
13.  main
```

E.g., the name ParseContext suggests that an object is temporary/local to the parsing operation.

"**Learning-based Memory Allocation for C++ Server Workloads**", ASPLOS 2020, Martin Maas, David G. Andersen, Michael Isard, Mohammad Mahdi Javanmard, Kathryn S. McKinley, Colin Raffel

The LLAMA Allocator

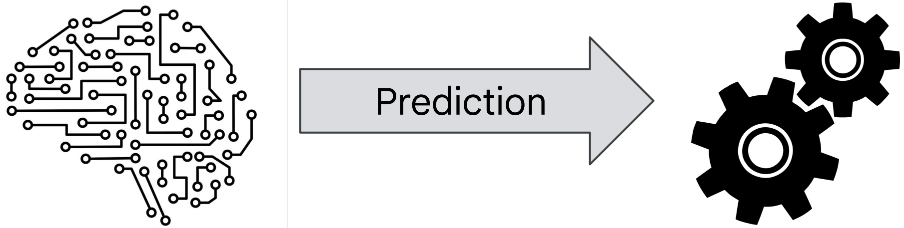


Pack objects with the same predicted lifetime into the same regions and fill gaps with shorter-lived objects

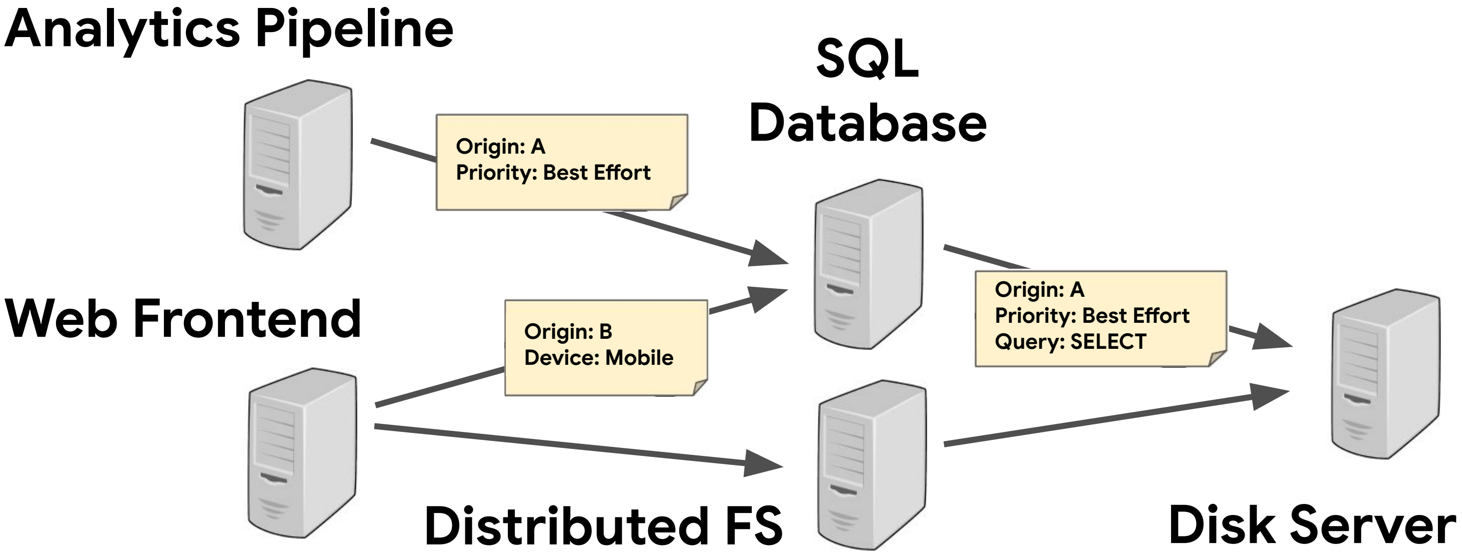
Allocator can detect and adapt to model mispredictions

"**Learning-based Memory Allocation for C++ Server Workloads**", ASPLOS 2020, Martin Maas, David G. Andersen, Michael Isard, Mohammad Mahdi Javanmard, Kathryn S. McKinley, Colin Raffel

Recipe: Separate a policy into a predictor and an algorithm that can tolerate errors.



Generalizes to Storage Systems

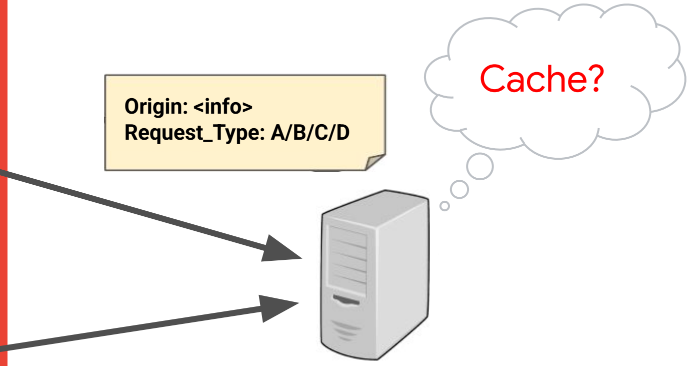


Metadata attached to storage requests helps predict behavior.

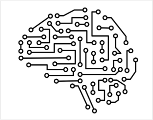
"Learning on Distributed Traces for Data Center Storage Systems", MLSys 2021, Giulio Zhou, Martin Maas

Google

Storage Prediction Tasks

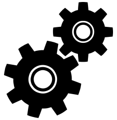


Disk Server



Predictable Property:

- Interarrival Times (Caching)

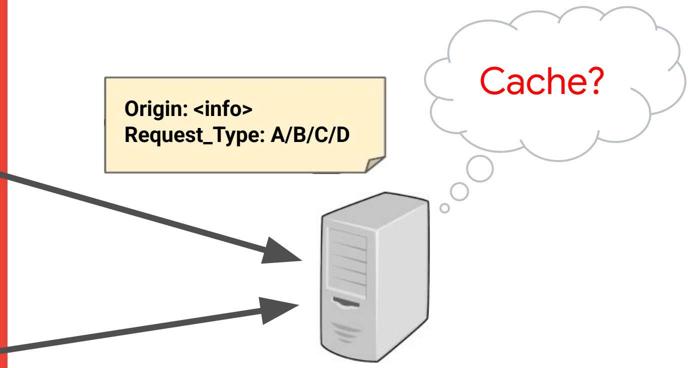


Algorithms:

- Cache admission and eviction.

"Learning on Distributed Traces for Data Center Storage Systems", MLSys 2021, Giulio Zhou, Martin Maas

Storage Prediction Tasks



Disk Server

Predictable Property:

- Interarrival Times (Caching)
- File Lifetime
- Final File Size
- Read/Write Ratio

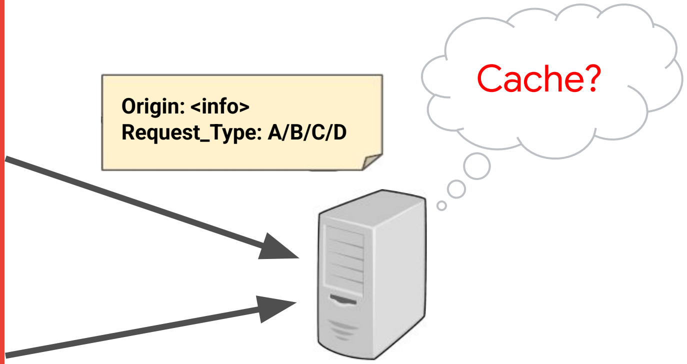
Algorithms:

- Cache admission and eviction.
- Place data on SSD vs. HDD.

"Learning on Distributed Traces for Data Center Storage Systems", MLSys 2021, Giulio Zhou, Martin Maas

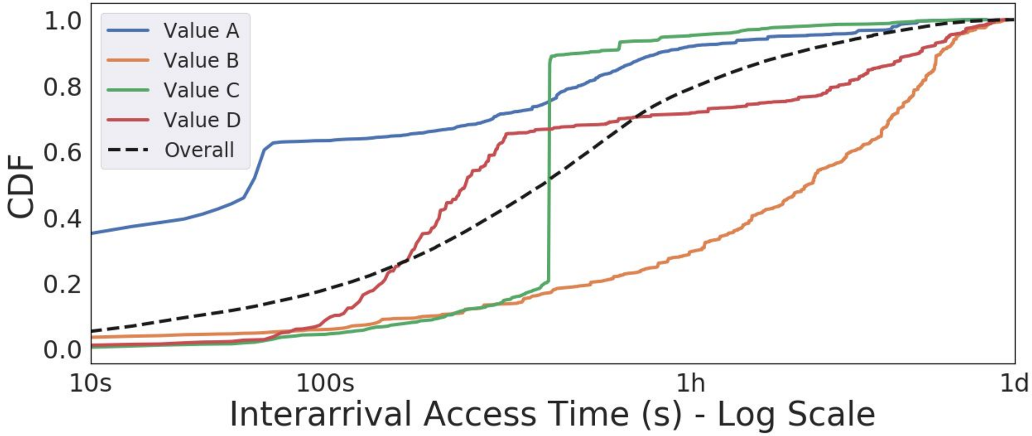
Challenge: Some properties are noisy and unpredictable.

Unpredictable Properties



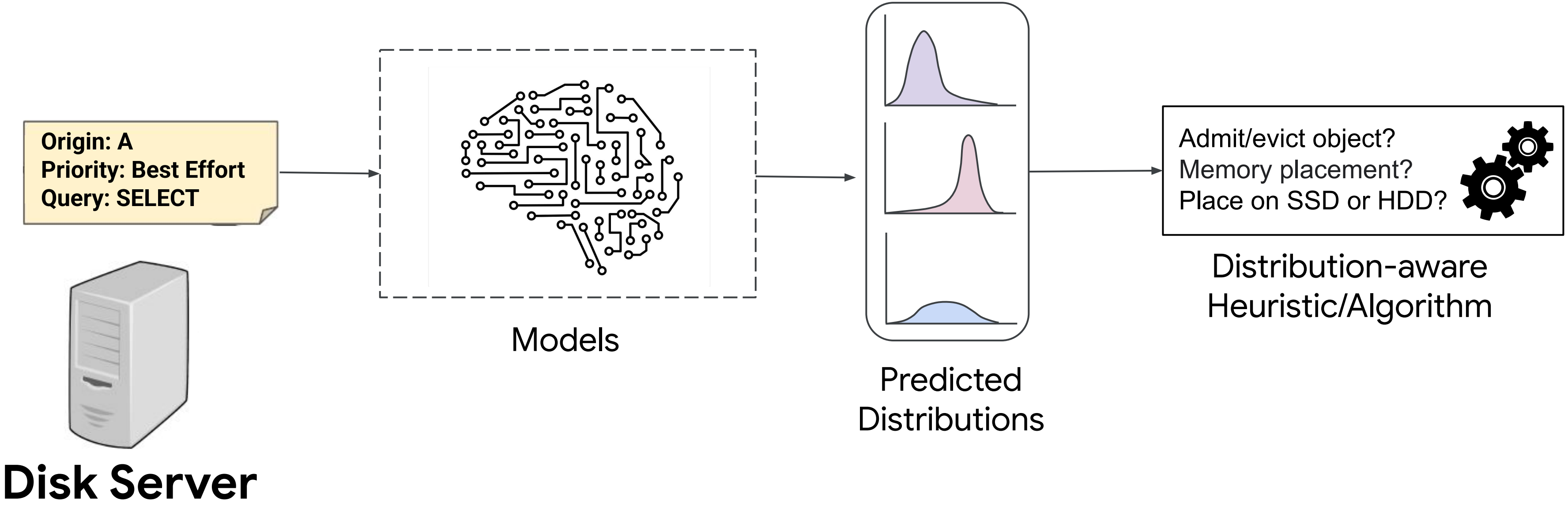
Disk Server

CDF by Request Type



"Learning on Distributed Traces for Data Center Storage Systems", MLSys 2021, Giulio Zhou, Martin Maas

Learning for Storage



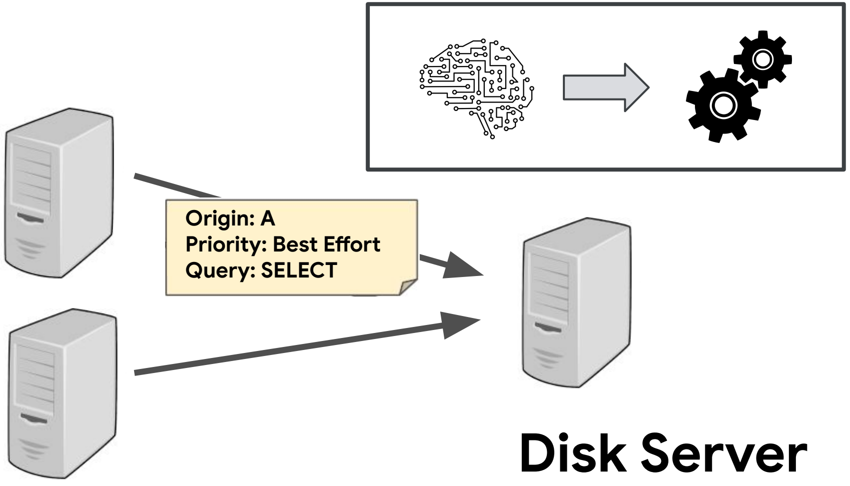
"Learning on Distributed Traces for Data Center Storage Systems", MLSys 2021, Giulio Zhou, Martin Maas

Recipe: Predict Distributions instead of specific values.

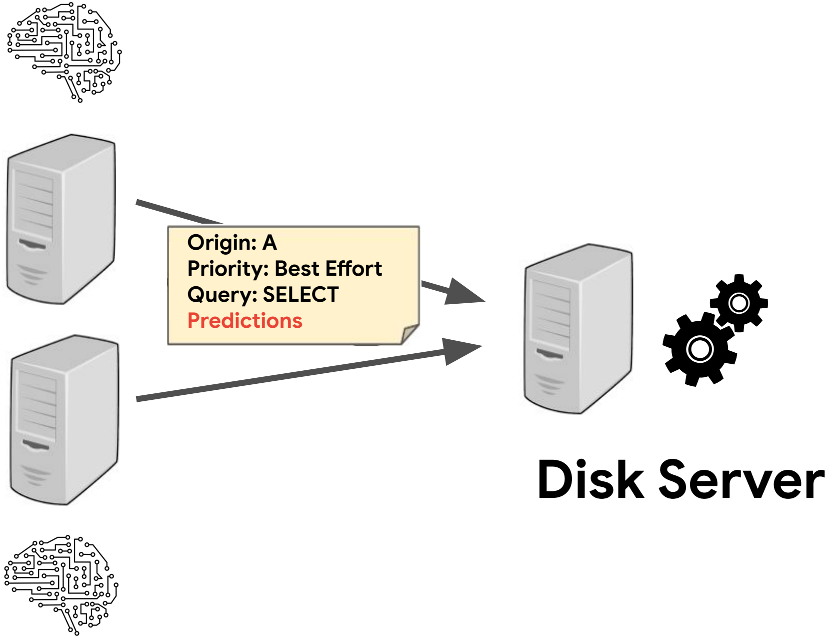
Challenge: What if the workloads shift over time?

"Bring Your Own Model"

Model in the system



Model in the workload



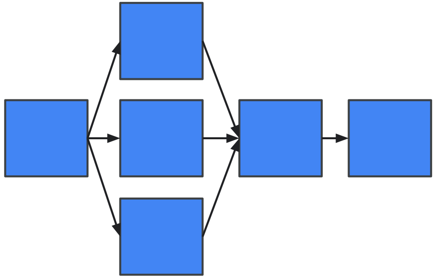
"A Bring-Your-Own-Model Approach for ML-Driven Storage Placement in Warehouse-Scale Computers", MLSys 2025, Chenxi Yang, Yan Li, Martin Maas, Mustafa Uysal, Ubaid Ullah Hafeez, Arif Merchant, Richard McDougall

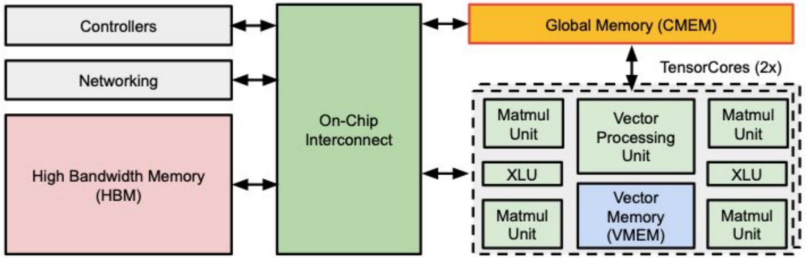
Recipe: Move the model into the workload.

Challenge: Some problems are very sensitive to errors.

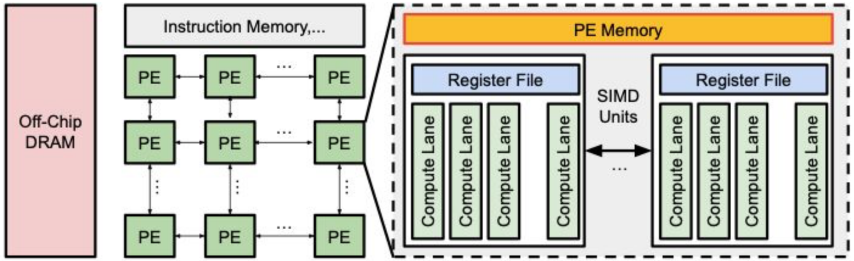
ML Accelerator Compilation

Memory Allocation: Take buffers with known start and end times and place them in a location within on-chip memory.





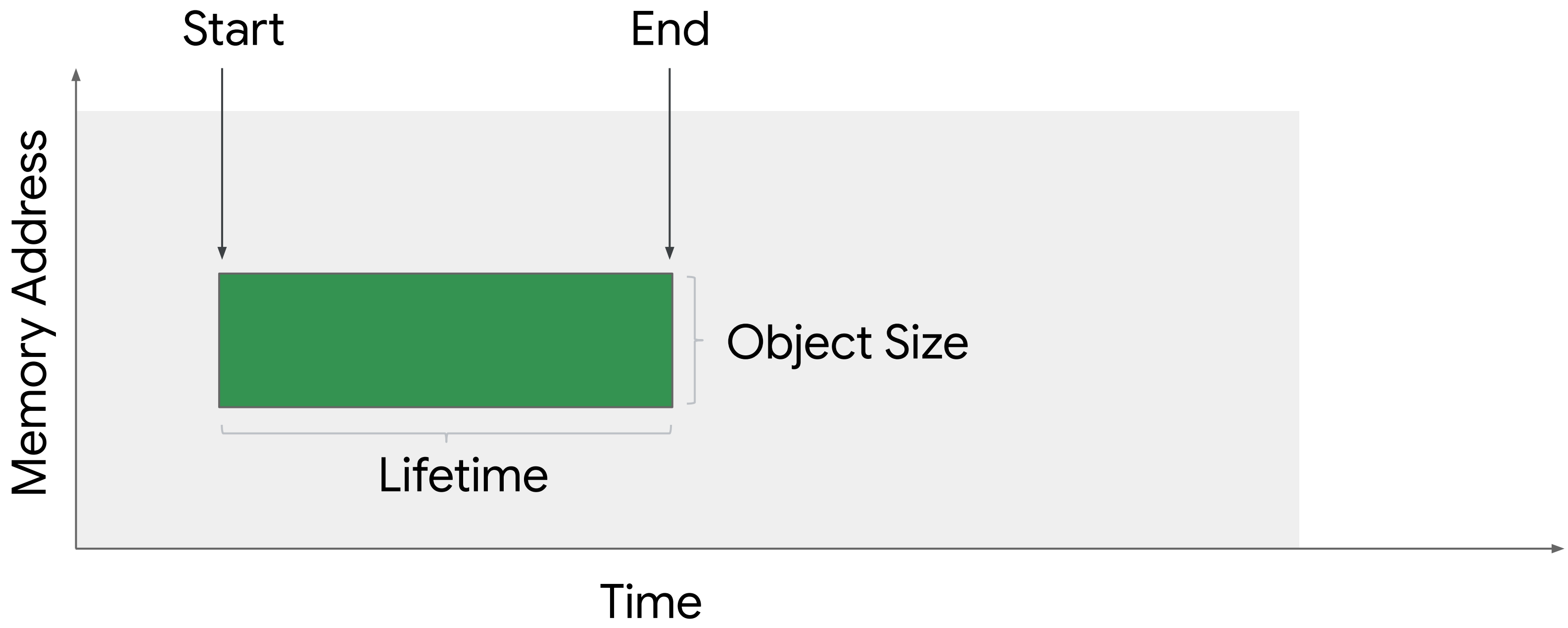
TPUv4



Pixel 6 Tensor SoC

"TelaMalloc: Efficient On-Chip Memory Allocation for Production Machine Learning Accelerators", ASPLOS 2023, Martin Maas, Ulysse Beaugnon, Arun Chauhan, Berkin Ilbeyi

High-Level Problem

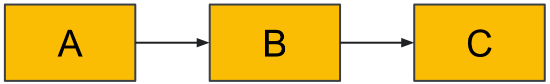


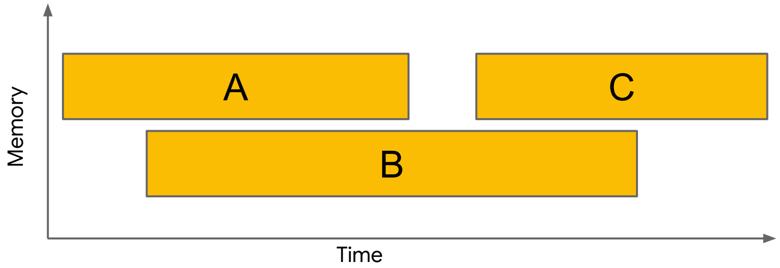
"TelaMalloc: Efficient On-Chip Memory Allocation for Production Machine Learning Accelerators", ASPLOS 2023, Martin Maas, Ulysse Beaugnon, Arun Chauhan, Berkin Ilbeyi

Google

Memory Allocation

Given a sequence of fixed-size buffers with a known start and end time, place them in memory such that total used memory never exceeds capacity.



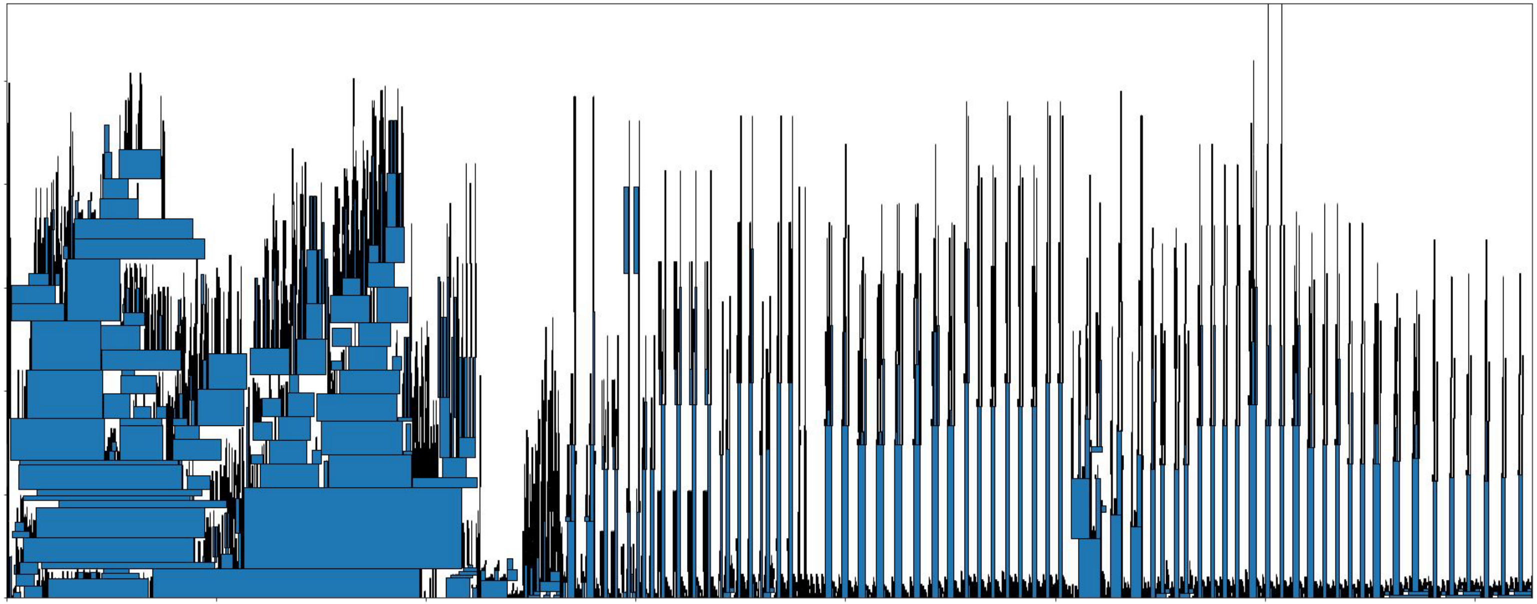


"TelaMalloc: Efficient On-Chip Memory Allocation for Production Machine Learning Accelerators", ASPLOS 2023, Martin Maas, Ulysse Beaugnon, Arun Chauhan, Berkin Ilbeyi

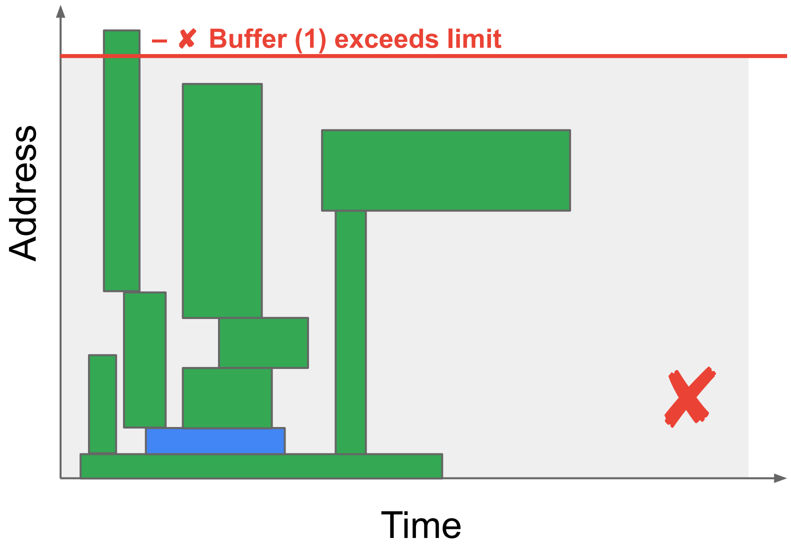
A Complex NP-Hard Problem

Heuristics: Fast, but do not always find a solution.

Solvers: Can handle complex inputs, but sometimes slow.

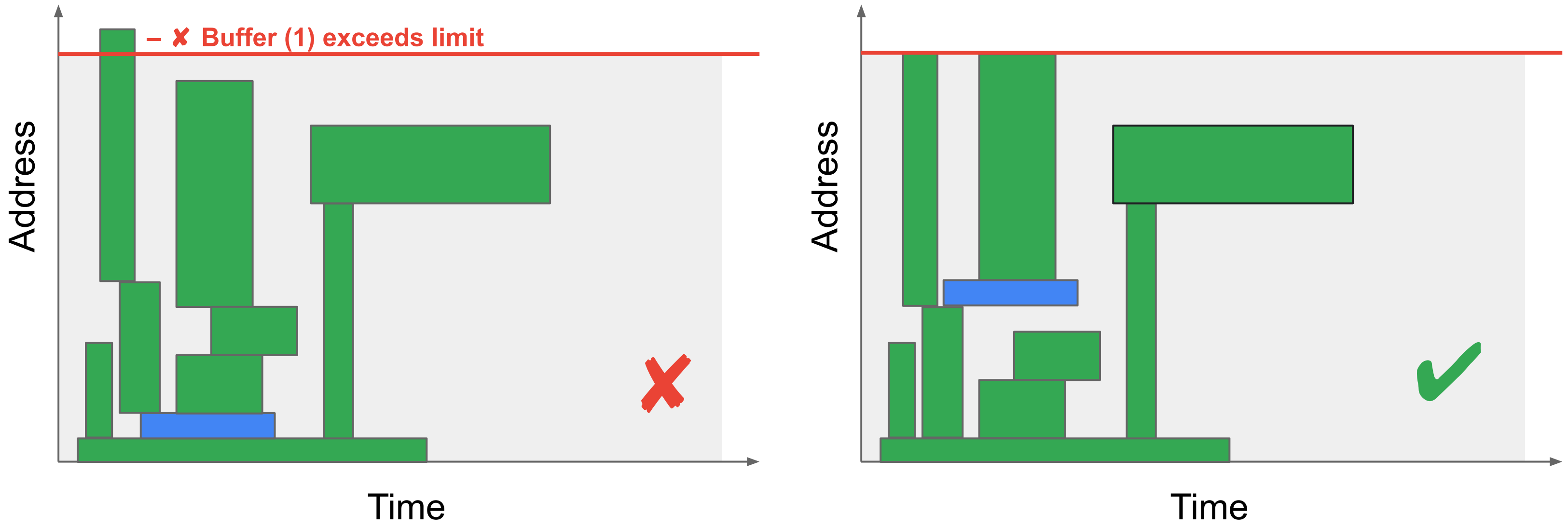


Limitations of Heuristics



"TelaMalloc: Efficient On-Chip Memory Allocation for Production Machine Learning Accelerators", ASPLOS 2023, Martin Maas, Ulysse Beaugnon, Arun Chauhan, Berkin Ilbeyi

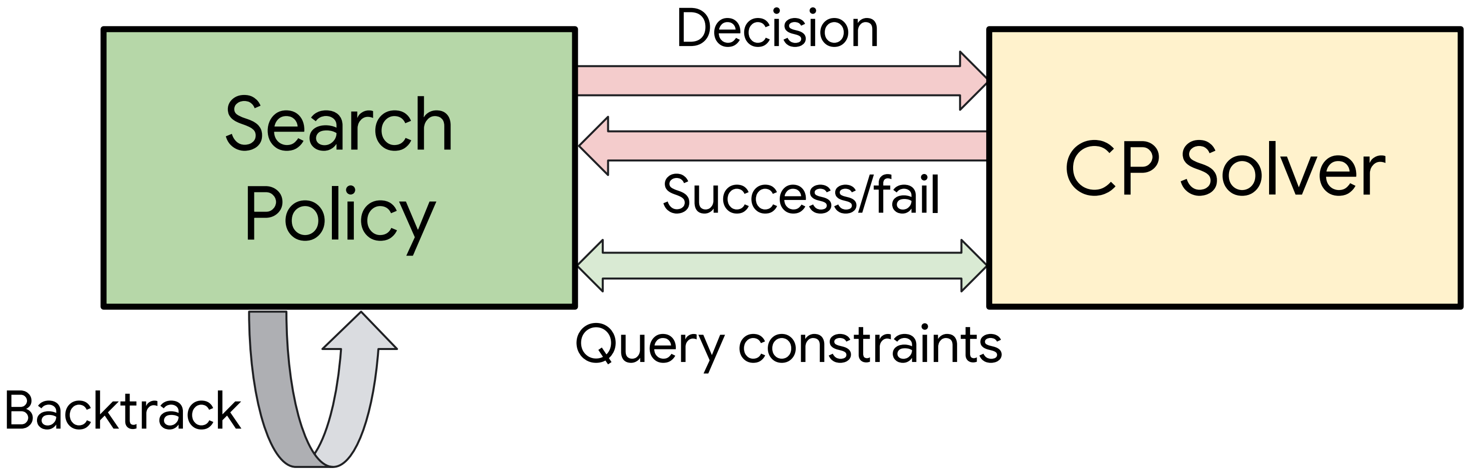
Limitations of Heuristics



"TelaMalloc: Efficient On-Chip Memory Allocation for Production Machine Learning Accelerators", ASPLOS 2023, Martin Maas, Ulysse Beaugnon, Arun Chauhan, Berkin Ilbeyi

Recipe: Enforce correctness by using ML to guide a solver.

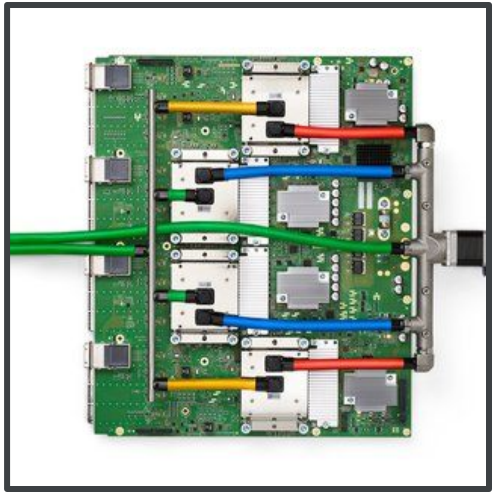
The Telamon Framework



“TelaMalloc: Efficient On-Chip Memory Allocation for Production Machine Learning Accelerators”, ASPLOS 2023, Martin Maas, Ulysse Beaugnon, Arun Chauhan, Berkin Ilbeyi

Shipping in Production

Pixel 6

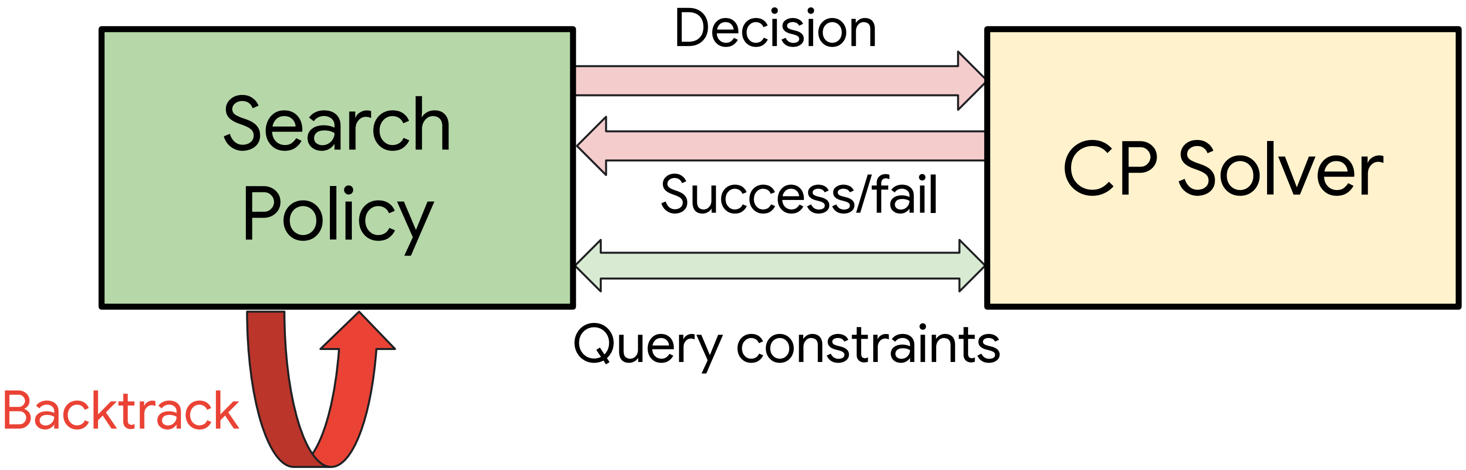


TPUv4

"TelaMalloc: Efficient On-Chip Memory Allocation for Production Machine Learning Accelerators", ASPLOS 2023, Martin Maas, Ulysse Beaugnon, Arun Chauhan, Berkin Ilbeyi

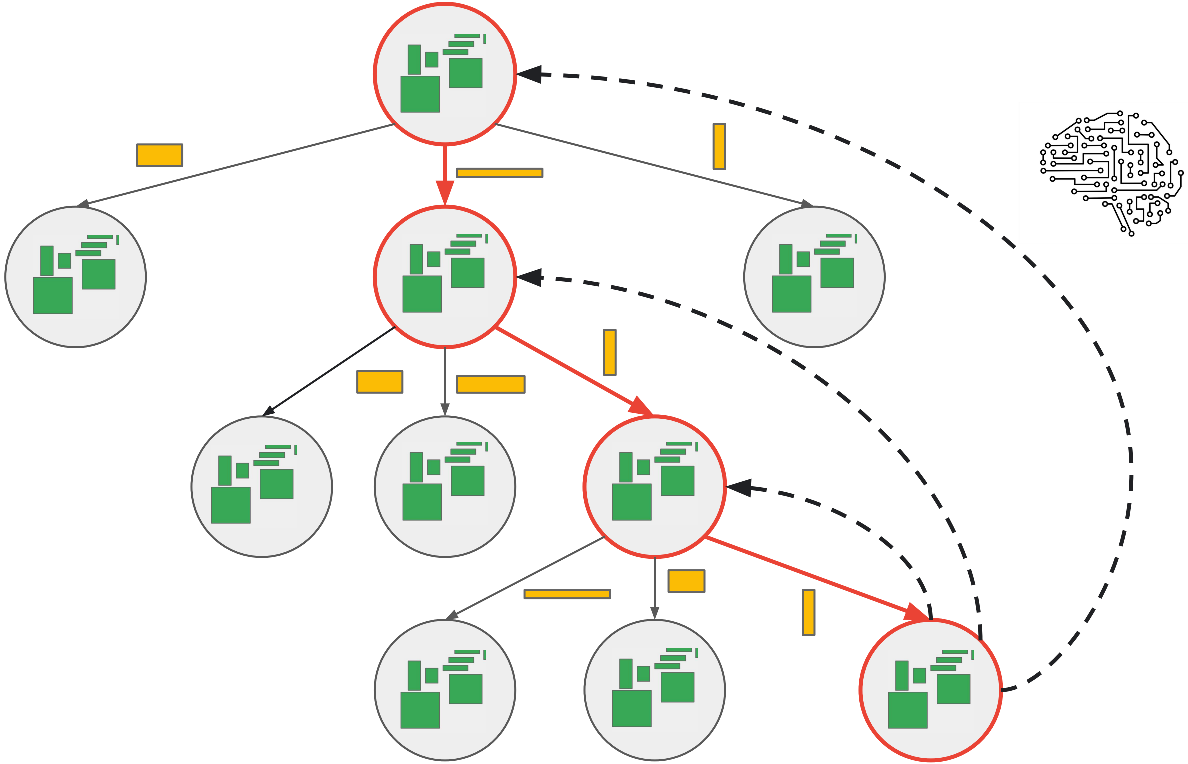
Google

Where can we apply ML?



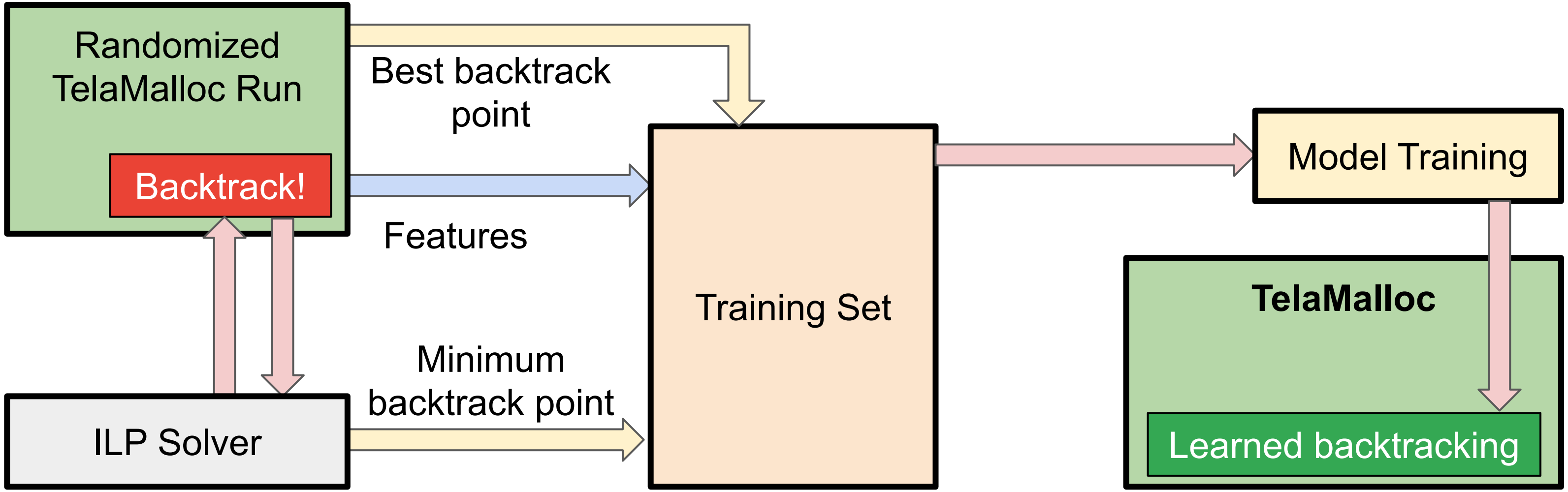
“**TelaMalloc: Efficient On-Chip Memory Allocation for Production Machine Learning Accelerators”,** ASPLOS 2023, Martin Maas, Ulysse Beaugnon, Arun Chauhan, Berkin Ilbeyi

The Backtracking Problem



How far to backtrack?

Imitation Learning



"TelaMalloc: Efficient On-Chip Memory Allocation for Production Machine Learning Accelerators", ASPLOS 2023, Martin Maas, Ulysse Beaugnon, Arun Chauhan, Berkin Ilbeyi

Recipe: Learn a heuristic using imitation learning.

Recap: ML for Systems Recipes

We have just seen five different **ML for Systems recipes**:

- Split policies into prediction + error-tolerant algorithm.
- Predict distributions for noisy data.
- Move the models out of the system if drift is high.
- Combine ML with a solver if high precision is required.
- Use imitation learning if problems can be solved offline.

This is only a small subset!

Catalyst #1:
Critical mass of generalizable design recipes.

Talk Outline

1 Conceptual Abstractions

Standardized ways for building learning into systems

2 ML Support in Systems

Best practices for deploying learning-based systems

3 Growing AI Capabilities

GenAI and other approaches

Talk Outline

1 Conceptual Abstractions

Standardized ways for building learning into systems

2 ML Support in Systems

Best practices for deploying learning-based systems

3 Growing AI Capabilities

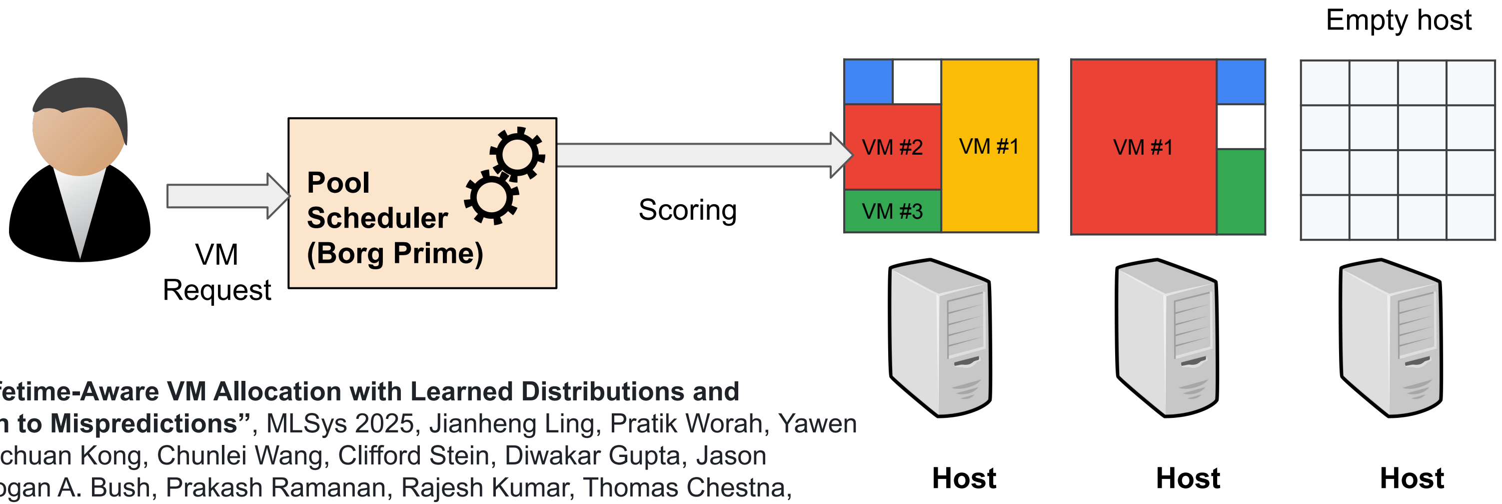
GenAI and other approaches

A common perception is that using ML in low-level systems is not practical.

Current systems are not designed to use ML, just like systems were not always designed for multi-threading.

Cluster Scheduling at Google

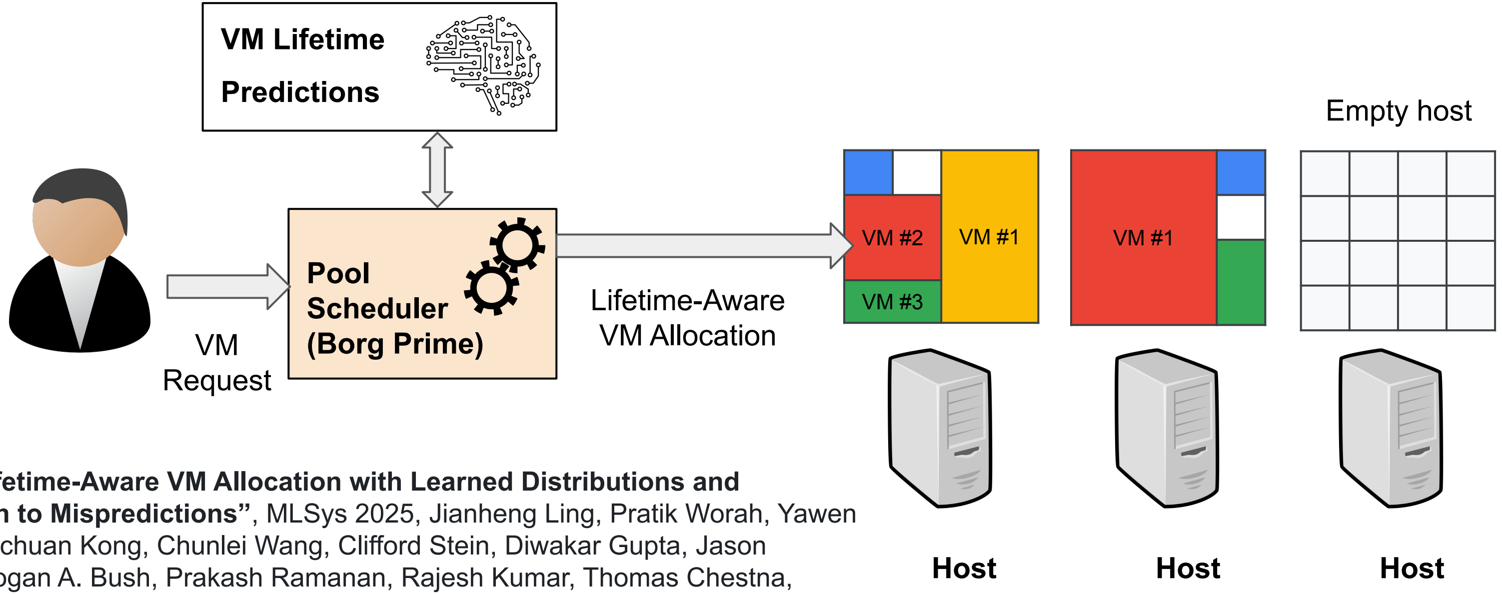
Building ML into VM allocation for Google Compute Engine (GCE). 100s to 10,000+ hosts per cluster, 10-100 scheduling request/second per cluster,



"LAVA: Lifetime-Aware VM Allocation with Learned Distributions and Adaptation to Mispredictions", MLSys 2025, Jianheng Ling, Pratik Worah, Yawen Wang, Yunchuan Kong, Chunlei Wang, Clifford Stein, Diwakar Gupta, Jason Behmer, Logan A. Bush, Prakash Ramanan, Rajesh Kumar, Thomas Chestna, Yajing Liu, Ying Liu, Ye Zhao, Kathryn S. McKinley, Meeyoung Park, Martin Maas

The LAVA Approach

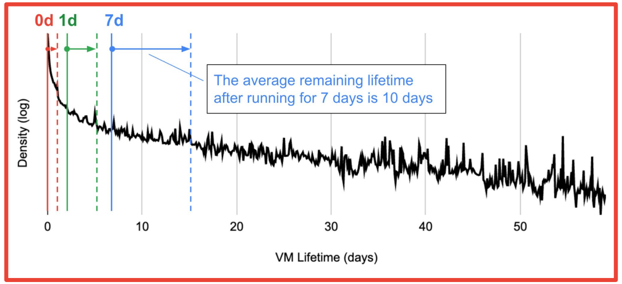
An instance of the **predictor+algorithm recipe** applied to **VM allocation**. Approach: **L**ifetime-**A**ware **V**M **A**llocation.

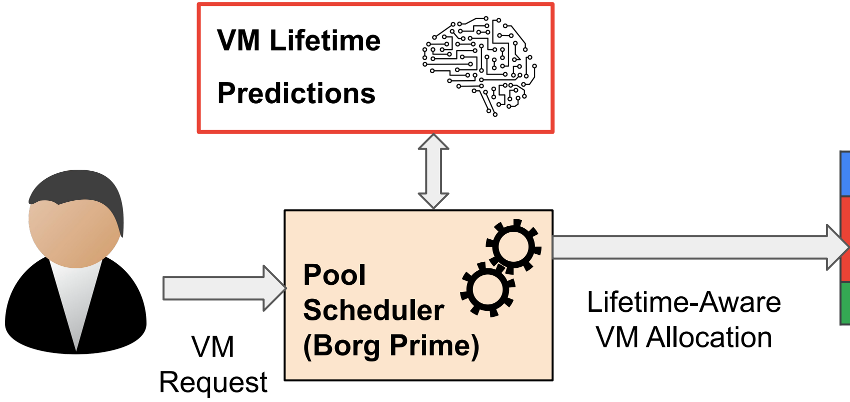


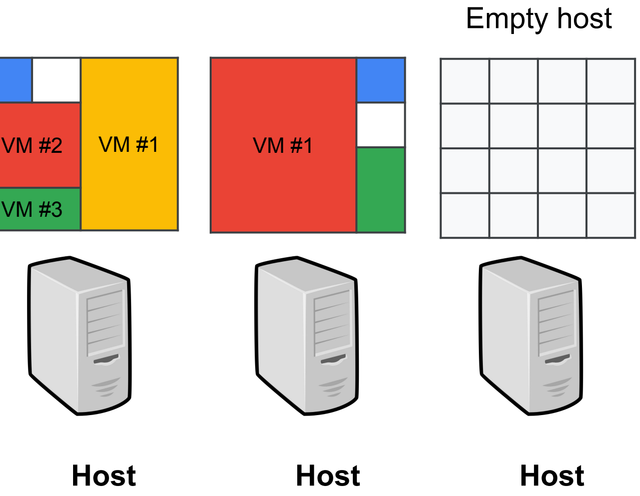
"LAVA: Lifetime-Aware VM Allocation with Learned Distributions and Adaptation to Mispredictions", MLSys 2025, Jianheng Ling, Pratik Worah, Yawen Wang, Yunchuan Kong, Chunlei Wang, Clifford Stein, Diwakar Gupta, Jason Behmer, Logan A. Bush, Prakash Ramanan, Rajesh Kumar, Thomas Chestna, Yajing Liu, Ying Liu, Ye Zhao, Kathryn S. McKinley, Meeyoung Park, Martin Maas

The LAVA Approach

Distribution-based prediction with repredictions over time.



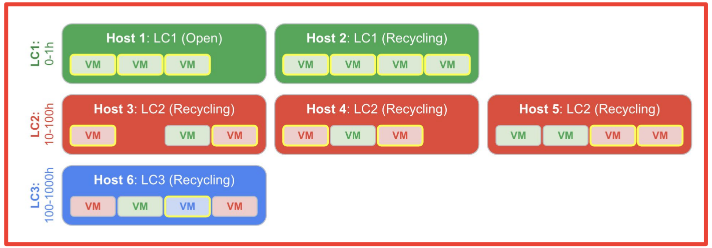


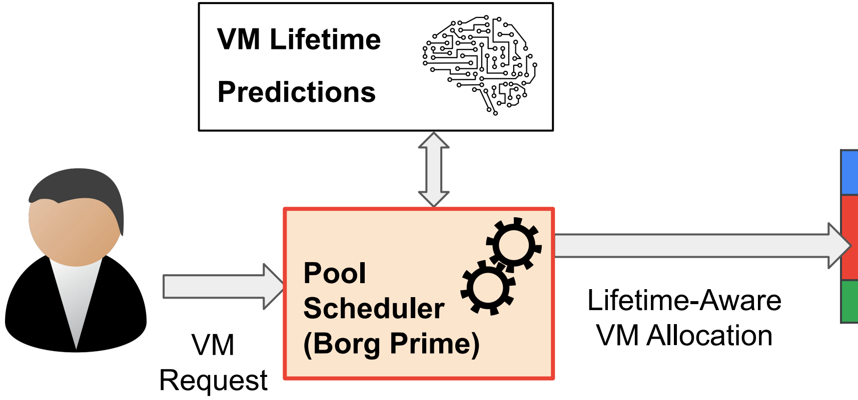


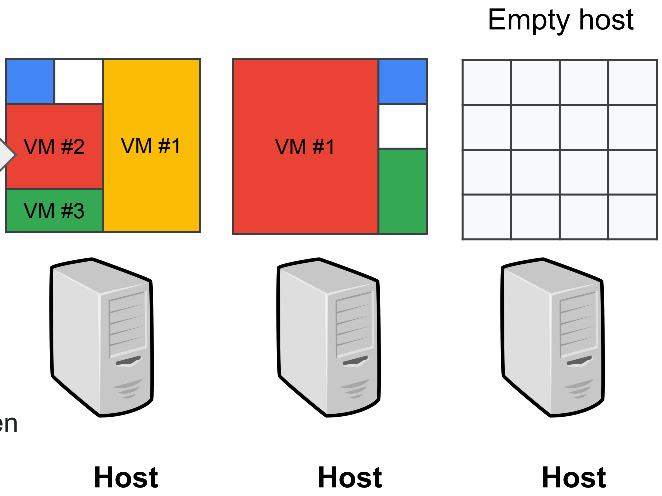
"LAVA: Lifetime-Aware VM Allocation with Learned Distributions and Adaptation to Mispredictions", MLSys 2025, Jianheng Ling, Pratik Worah, Yawen Wang, Yunchuan Kong, Chunlei Wang, Clifford Stein, Diwakar Gupta, Jason Behmer, Logan A. Bush, Prakash Ramanan, Rajesh Kumar, Thomas Chestna, Yajing Liu, Ying Liu, Ye Zhao, Kathryn S. McKinley, Meeyoung Park, Martin Maas

The LAVA Approach

Scheduling algorithm corrects for mispredictions, similar to LLAMA.







"LAVA: Lifetime-Aware VM Allocation with Learned Distributions and Adaptation to Mispredictions", MLSys 2025, Jianheng Ling, Pratik Worah, Yawen Wang, Yunchuan Kong, Chunlei Wang, Clifford Stein, Diwakar Gupta, Jason Behmer, Logan A. Bush, Prakash Ramanan, Rajesh Kumar, Thomas Chestna, Yajing Liu, Ying Liu, Ye Zhao, Kathryn S. McKinley, Meeyoung Park, Martin Maas

Deploying ML in Systems

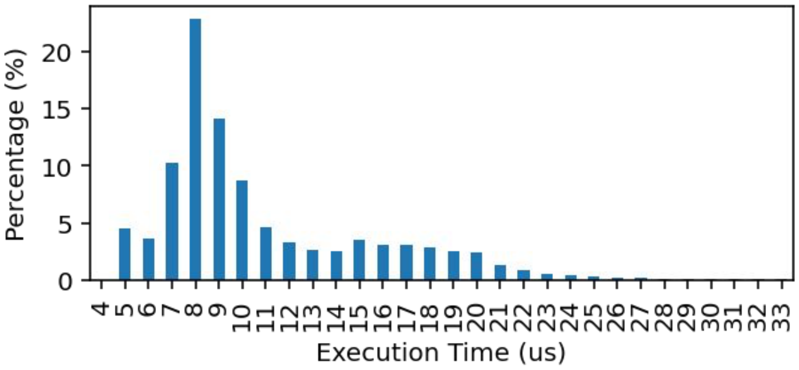
Deploying learning-based systems has unique challenges, which we expect to **generalize** across different settings:

- Latency Constraints.
- Safety and correctness guarantees.
- Integration into rollout processes.
- Explainability and interpretability.

Prediction Latency

Prediction is **on the critical path** for VM scheduling and may involve re-scoring O(10-100) VMs per request.

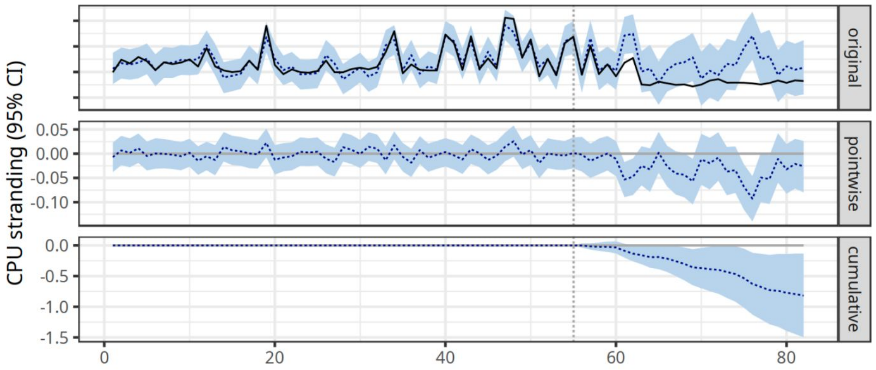
- Gradient boosted trees.
- Run within the scheduler itself, not on other servers.
- Median latency is **9 us**.



Safety & Correctness

Model stays up-to-date for months, is updated regularly.

- **Offline training** with careful **backtesting**.
- Model is subject to **testing** and **gradual roll-out**.
- Careful **monitoring** for production regressions.
- Ran **pilots** with causal analysis and A/B testing.



Rollout Processes

Option 1: Roll out the model independently of the system.

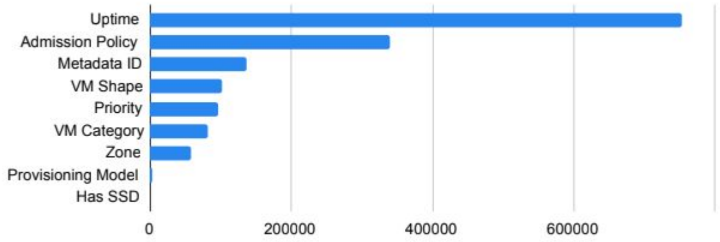
- ✔ Allows updating model more often than the system.
- ✘ Might break verification assumptions.

Option 2: Roll out model with the system binary. **(Ours)**

- ✔ Can leverage existing rollout testing.
- ✘ Model might be stale when it reaches production.

Explainability/Interpretability

- Model the problem in a way that it becomes **naturally interpretable** (e.g., predictor+algorithm recipe).
- Use explainable model libraries (e.g., decision trees).
- Use interpretability techniques (e.g., TCAV).



Feature importance for LAVA models, calculated by the gradient boosted tree library.

We had to build a lot of custom infrastructure and change our systems to integrate ML.

Building **general** systems that enable such approaches is a great research opportunity for academia and industry.

There is sometimes a **perception** that ML for Systems is difficult to work on in academia.

Myths about ML for Systems

Myth #1: ML for Systems needs lots of data.

- Academia-scale system deployments often produce sufficient amounts of data.
- Among ML application areas, this might make ML for Systems particularly well-suited for academia.
- The lifetime-based memory allocation paper could have been done in academia.

Myths about ML for Systems

Myth #2: ML for Systems needs lots of compute.

- A lot of work can be done without fine-tuning or other expensive model training.
- Cheap models (e.g., random forests) are often enough.
- Can build on publicly available models, in the cloud or open-weight models (e.g., Gemma).
- Leverage in-context learning instead of training.

Myths about ML for Systems

Myth #3: I need to be a Machine Learning Expert.

- Many problems in this space are systems problems.
- Can often treat ML as opaque building block.
- LLMs further reduce the learning curve.

Example: ML-Enabled Systems

Lots of work in the community (an incomplete list):

- **LAKE**: ML-enabled OS Kernel.
  "Towards a Machine Learning-Assisted Kernel with LAKE", Henrique Fingler, et al., ASPLOS'23.
- **ArchGym**: Environment for ML for hardware.
  "ArchGym: An Open-Source Gymnasium for Machine Learning Assisted Architecture Design", Srivatsan Krishnan, et al., ISCA'23.
- **DeathStarBench**/Sage/Seer: ML for microservices.
  "An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud & Edge Systems", Yu Gan, et al., ASPLOS'19.
- **MLGO**: Integrating ML into LLVM.
  "MLGO: a Machine Learning Guided Compiler Optimizations Framework", Mircea Trofin, et al., arXiv:2101.04808.
- **Scalene**: Integrating ML into performance profiling.
  "Triangulating python performance issues with SCALENE", Emery Berger, et al., OSDI'23.

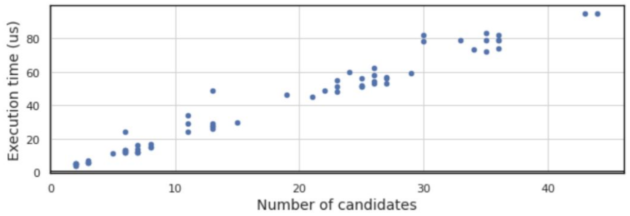
Example: ML Libraries

Easy-to-deploy random forest library:

- Trains from the command line.
- Integrates into C++, Python binaries, etc. at very low latency.
- Can ship models with a binary or load them from a file system.
- Explainability support built in.

YDF

Yggdrasil Decision Forests



Latency of Telamon models: 2us/sample

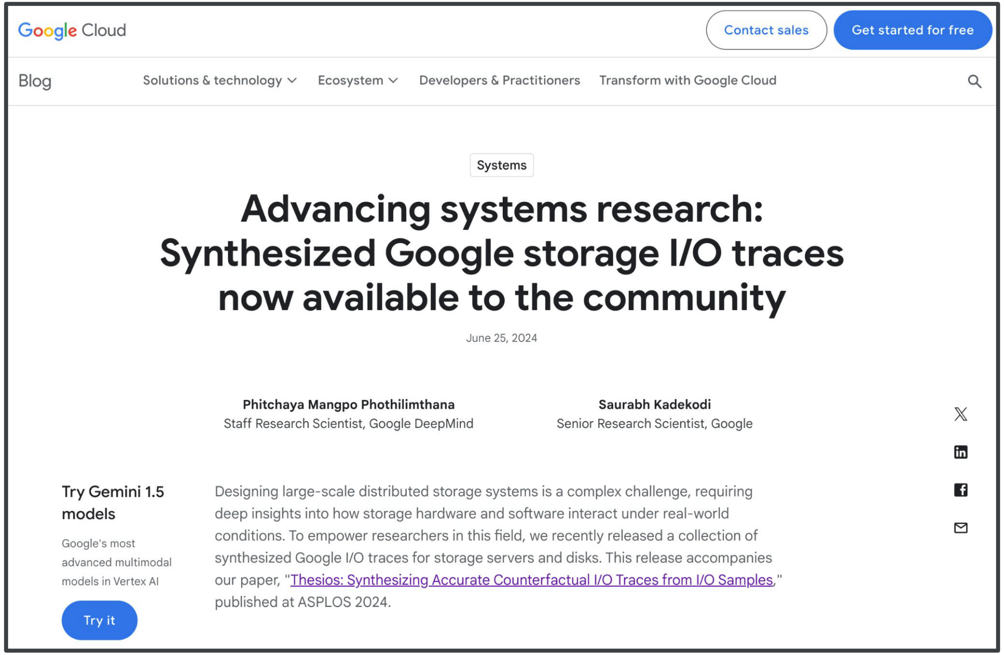
Yggdrasil Decision Forests: A Fast and Extensible Decision Forests Library, Guillame-Bert et al., KDD 2023

Example: HW/SW Co-Design

Opportunities for academic research on HW/SW co-design for learning-based systems.

- What hardware extensions would facilitate machine learning in low levels of the stack?
- The emergence of the open **RISC-V Instruction Set** greatly facilitates this kind of work:
    - Enables research using real hardware designs.
    - An opportunity to influence a real ISA?

Example: Industry Datasets



Google Cloud

Contact sales | Get started for free

Blog | Solutions & technology | Ecosystem | Developers & Practitioners | Transform with Google Cloud

Systems

Advancing systems research: Synthesized Google storage I/O traces now available to the community

June 25, 2024

Phitchaya Mangpo Phothilimthana
Staff Research Scientist, Google DeepMind

Saurabh Kadekodi
Senior Research Scientist, Google

Try Gemini 1.5 models

Google's most advanced multimodal models in Vertex AI

Try it

Designing large-scale distributed storage systems is a complex challenge, requiring deep insights into how storage hardware and software interact under real-world conditions. To empower researchers in this field, we recently released a collection of synthesized Google I/O traces for storage servers and disks. This release accompanies our paper, "Thesios: Synthesizing Accurate Counterfactual I/O Traces from I/O Samples," published at ASPLOS 2024.

2-month-long synthesized traces from **3 different Google storage clusters**, containing approximately **2.5 billion I/O records**.

https://github.com/google-research-datasets/thesios

"**Thesios: Synthesizing Accurate Counterfactual I/O Traces from I/O Samples**", ASPLOS 2024, Phitchaya Mangpo Phothilimthana, Saurabh Kadekodi, Soroush Ghodrati, Selene Moon, Martin Maas

Catalyst #2:

Systems and infrastructure that facilitate the integration of ML.

Talk Outline

1 Conceptual Abstractions
Standardized ways for building learning into systems

2 ML Support in Systems
Best practices for deploying learning-based systems

3 Growing AI Capabilities
GenAI and other approaches

Talk Outline

1 Conceptual Abstractions
Standardized ways for building learning into systems

2 ML Support in Systems
Best practices for deploying learning-based systems

3 Growing AI Capabilities
GenAI and other approaches

Over the past 5 years, we have seen a **rapid increase** in AI capabilities, especially GenAI.

The AI Capability Delta

Capability Delta: What are things that AI can do that we cannot possibly do with conventional approaches?

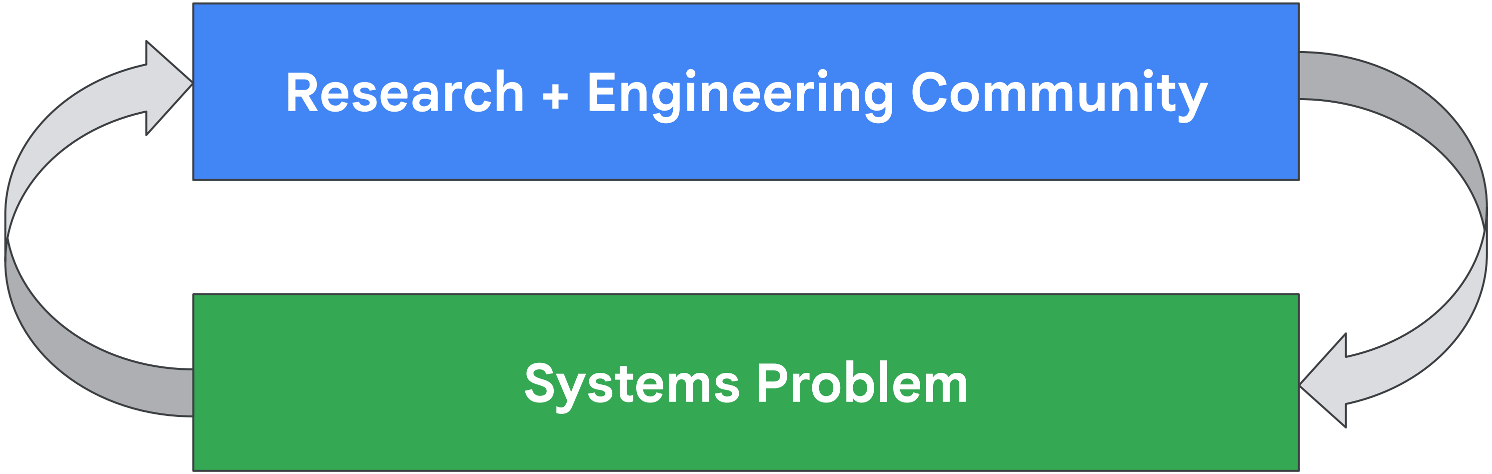
- In **image classification**, no non-learning technique achieved the performance of DNNs.
- No non-AI technique was able to **generate photorealistic images** like GANs and later GenAI.
- In the natural sciences, AlphaFold enabled **protein folding** at a scale that was previously impossible.

What is the Capability Delta for Systems?

What if the value of AI is not only to improve systems, but to **accelerate their evolution**?

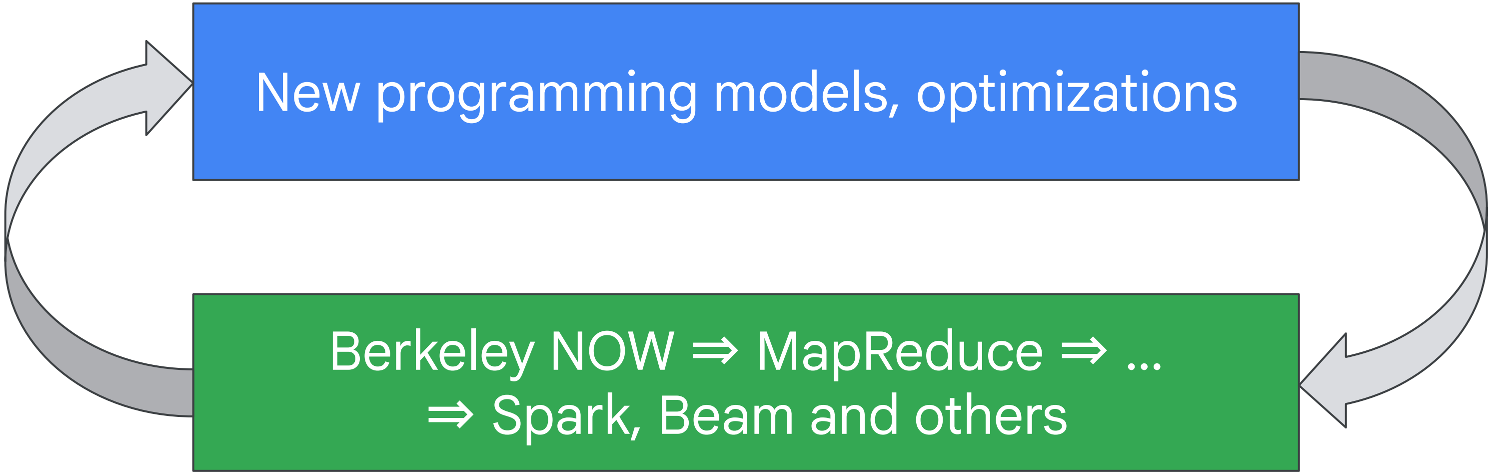
The System Optimization Loop

Once a systems problem is established, researchers spend years or decades to optimize solutions.



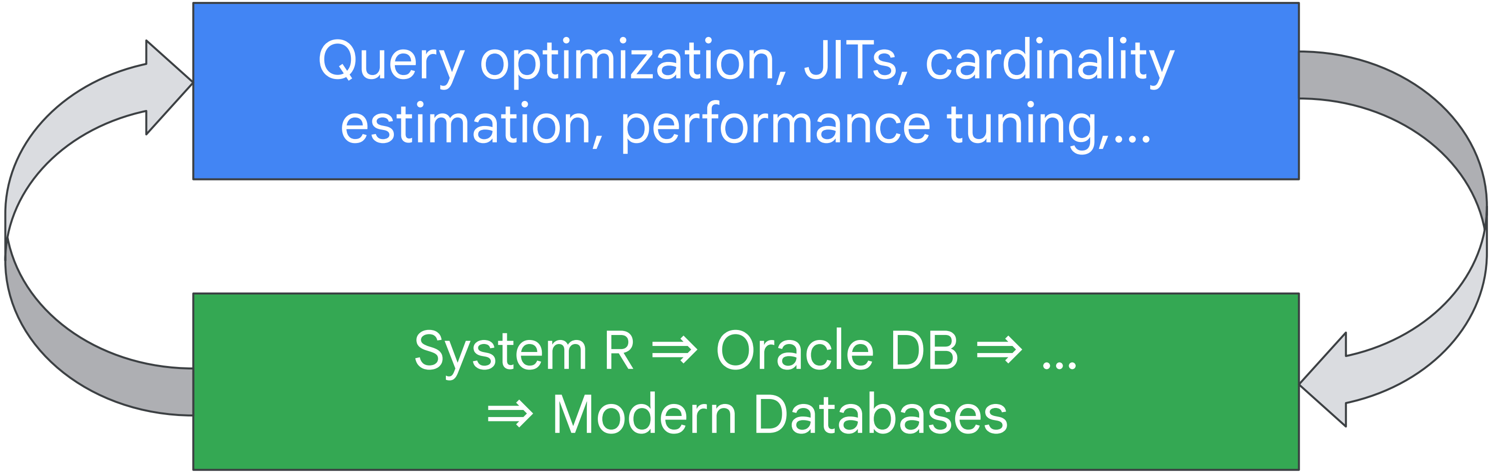
The System Optimization Loop

Once a systems problem is established, researchers spend years or decades to optimize solutions.



The System Optimization Loop

Once a systems problem is established, researchers spend years or decades to optimize solutions.



Automatic Optimization

Can AI improve the optimization loop of systems?

- **Quality**: Find optimizations a human did not find or would not have thought of.
- **Velocity**: Get to an efficient system more quickly.
- **Coverage**: Optimize systems that would otherwise not have enough support to be optimized.

ECO – Efficient Code Optimizer

Hannah Lin, Martin Maas, Maximilian Roquemore, Arman Hasanzadeh, Fred Lewis, Yusuf Simonson, Tzu-Wei Yang, Amir Yazdanbakhsh, Deniz Altinbüken, Florin Papa, Maggie Nolan Edmonds, Aditya Patil, Don Schwarz, Satish Chandra, Chris Kennelly, Milad Hashemi, Parthasarathy Ranganathan

More details: arXiv:2503.15669

Google

Optimizing Google's Code

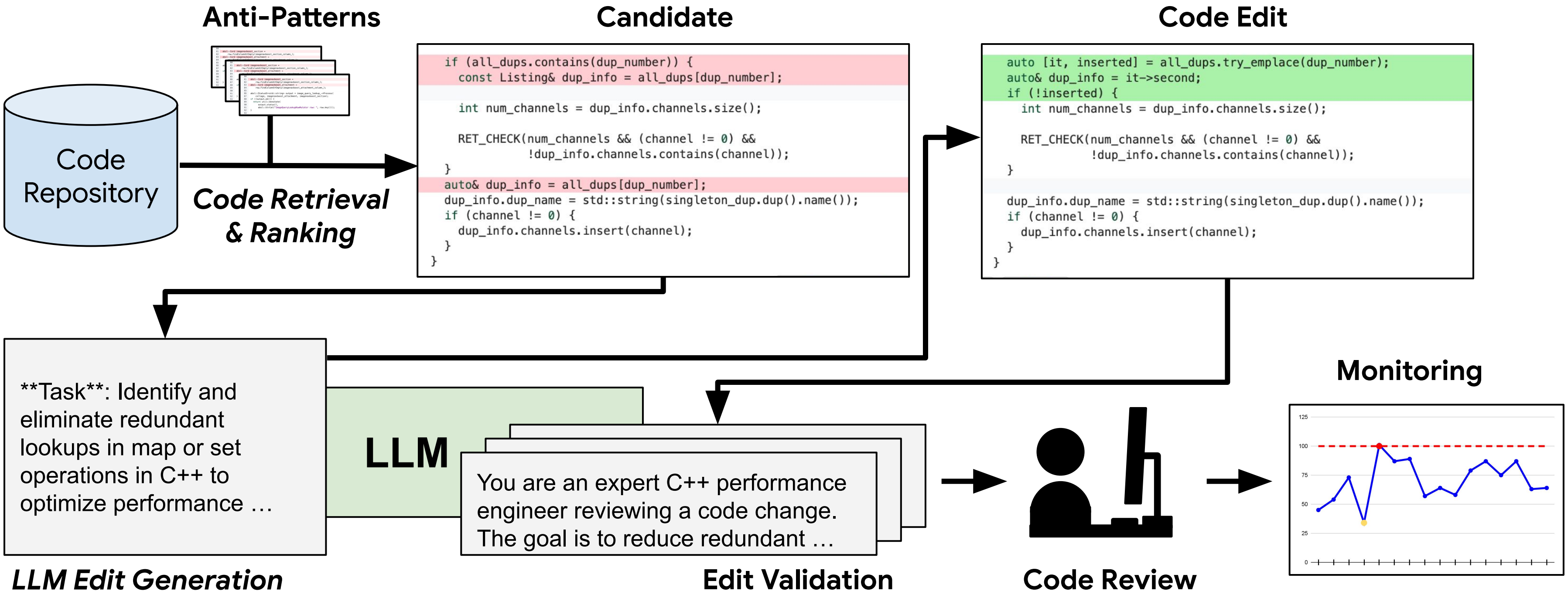
Google's Planet Scale Infrastructure:

- A global fleet of warehouse-scale computers.
- Mono-repo with **billions of lines of code**.
- Engineers spend a large amount of time optimizing code, including with automated tools.

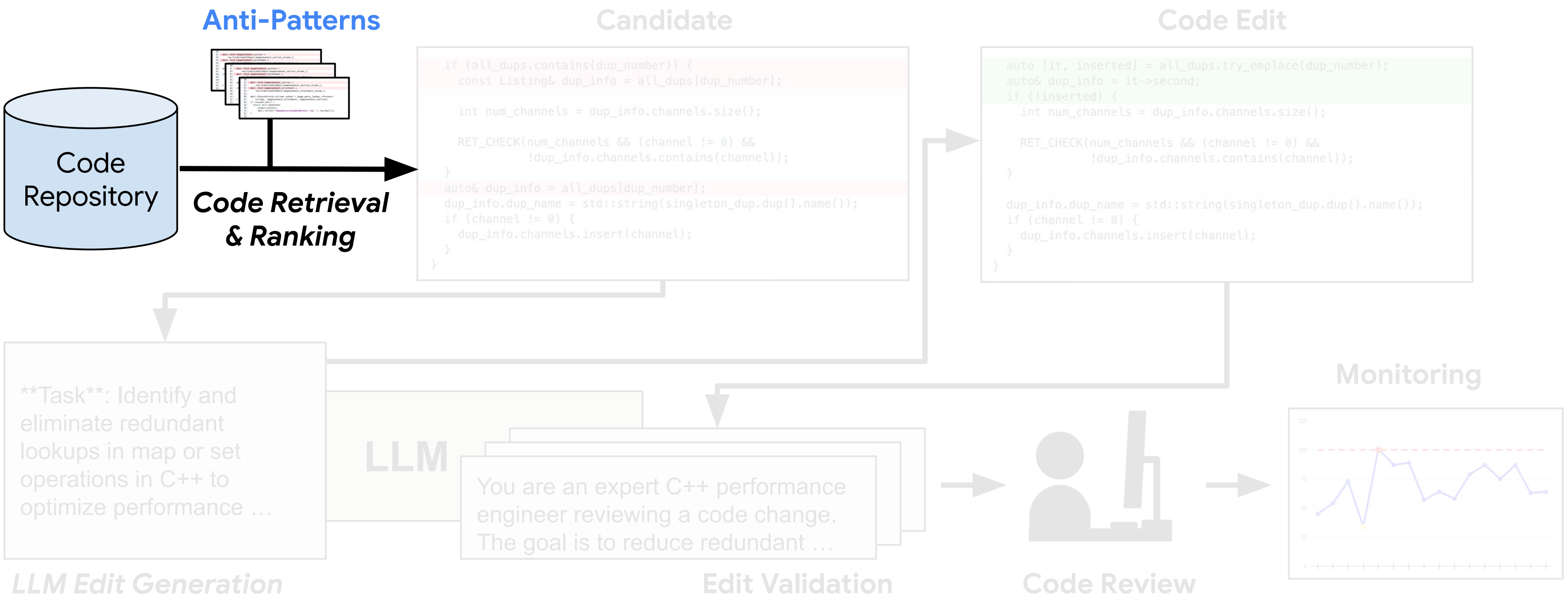
ECO uses AI to improve the optimization loop.

Potvin, R., & Levenberg, J. (2016). **Why Google stores billions of lines of code in a single repository**. Communications of the ACM, 59(7), 78-87.

Scaling Up Code Optimization



Scaling Up Code Optimization



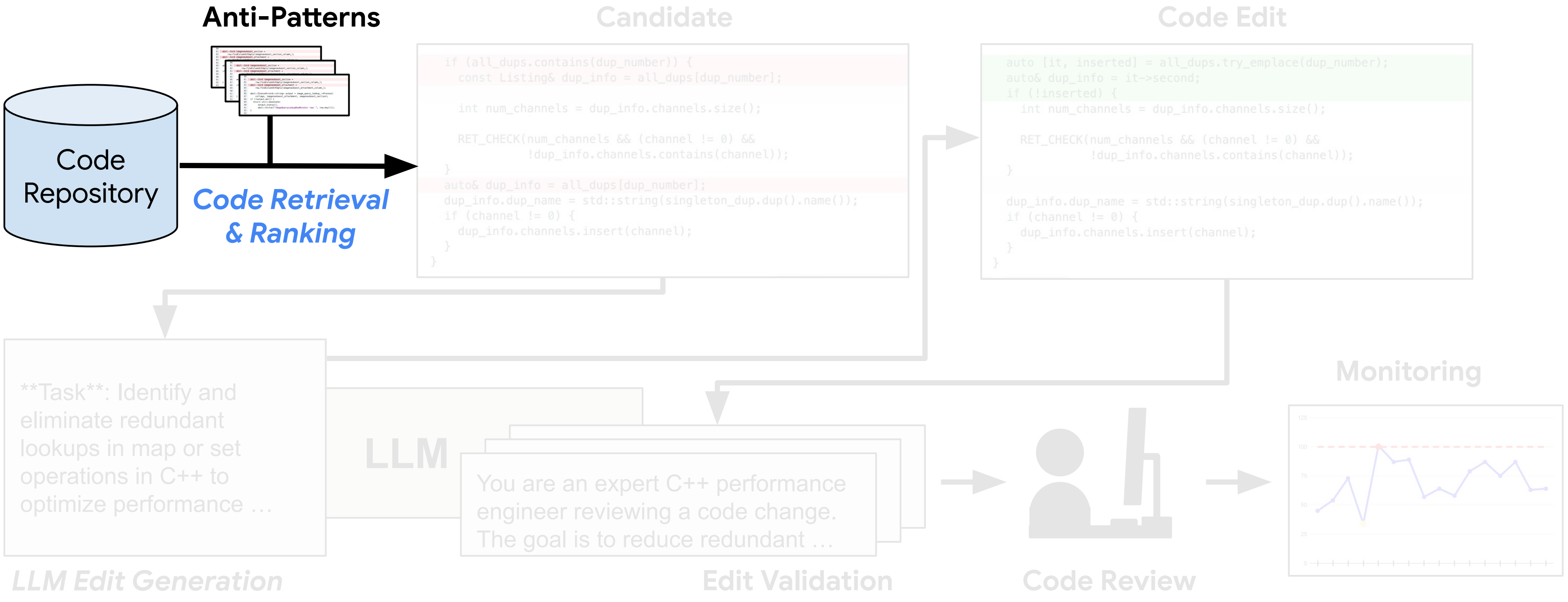
Code Edit Anti-Patterns

Find examples of performance-improving changes in the edit history of Google's large mono-repo.

- Static Analysis
- Annotations
- Keyword Search
- Documentation

Over **55K changes** stored in a database.

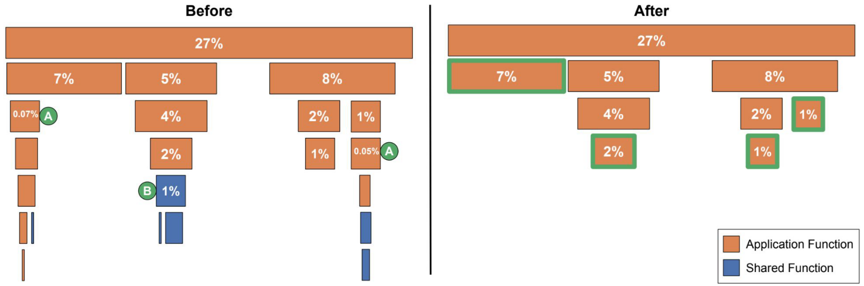
Scaling Up Code Optimization



Capturing Candidates

Use continuous profiling across our fleet and annotate functions with resource consumption.

- A large fraction of resources are in library functions (e.g., `vector::push_back`).
- Roll up resource to identify the most relevant functions.

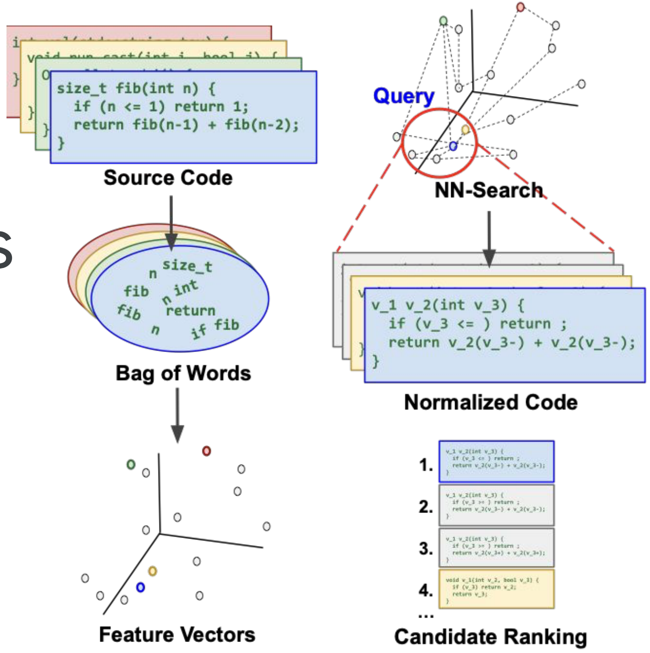


Code Retrieval

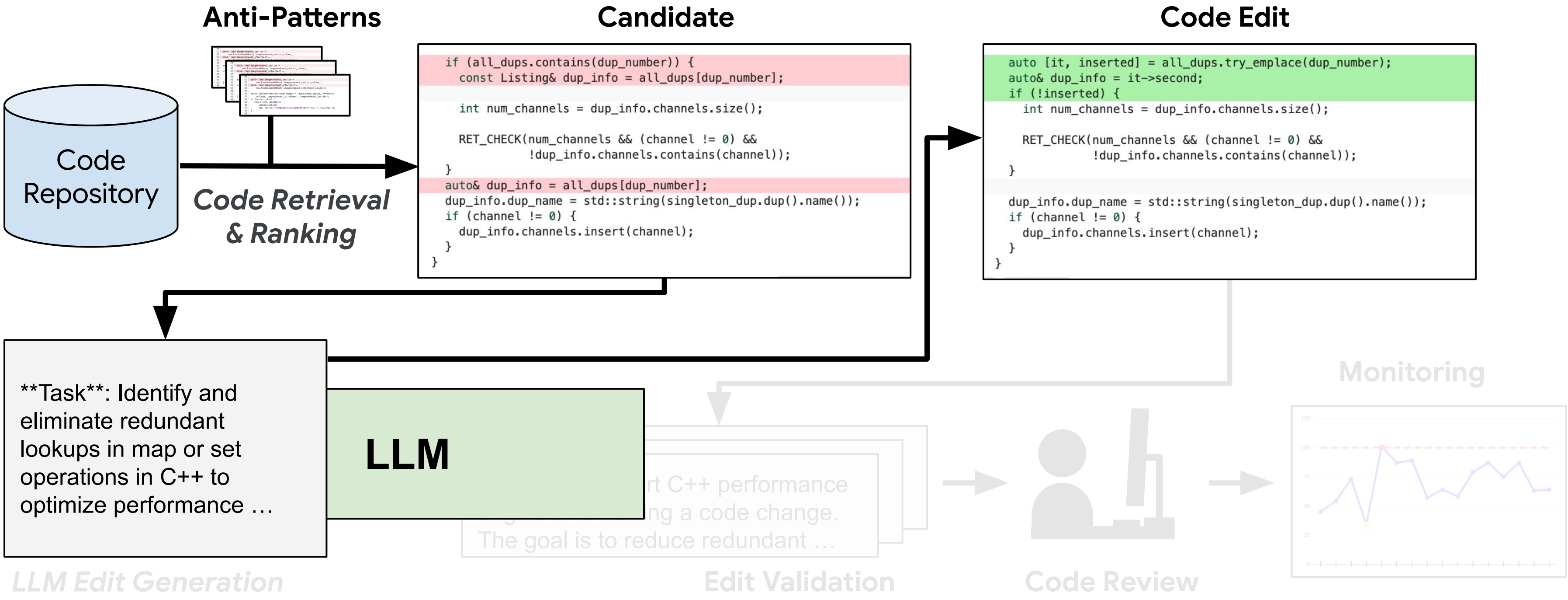
Encode candidates and store them in a vector database (ScaNN).

Used different embeddings (mappings from code into the vector space):

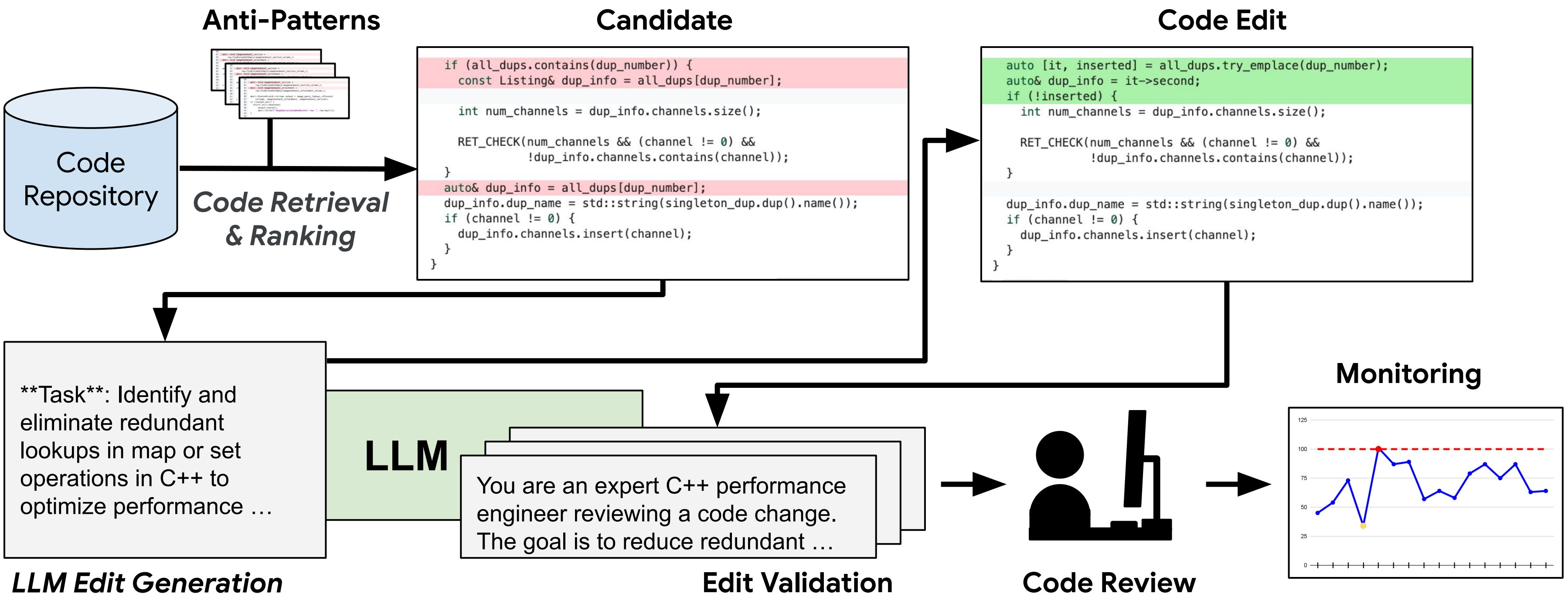
- Bag of words
- Deep embeddings



Scaling Up Code Optimization



Scaling Up Code Optimization



Validation & Code Review

Validate changes via LLM prompting and testing.

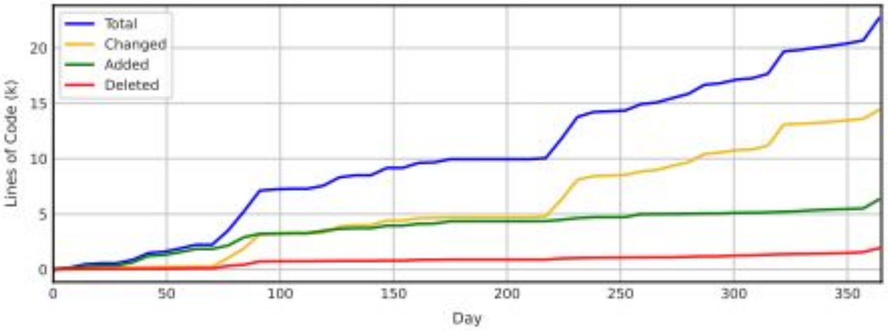
- Leverage automated testing and self-review to ensure changes are correct and high quality.
- Sent out to human reviewers for code review.
- After submission, monitor impact fleet-wide.

Note that the **human is still in the loop**, but we rapidly speed up and scale up the ability to optimize.

Impact

- Over **25K lines of code** changes submitted in 1 year.
- Saved the equivalent of 2M (normalized) CPU cores.
- **Less than 0.5% of changes had to be rolled back.**

	Copy	Map	Vector	Description
S_PROD	39.97%	4.86%	40.89%	Submitted to production.
S_USER	6.55%	10.49%	15.99%	Submitted with user discussion (e.g., reviewer questions or suggested changes).
R_REVERT	0.16%	0.14%	0.12%	Submitted to production but later rolled back due to regressions.
R_TEST	16.14%	37.39%	20.07%	Rejected during the validation phase.
R_USER	35.81%	46.55%	22.87%	Rejected by human reviewers.
R_EMPTY	0.00%	0.07%	0.00%	Rejected due to failure to make code modifications beyond formatting.
R_OTHER	1.37%	0.49%	0.05%	Rejected for other reasons, typically non-edit related, such as failure to resolve merge conflicts during submission or failure to identify proper reviewers.
Total Commits	4959	1421	4035	Total number of code commits that were generated.
NC Savings	331k	125k	618k	Efficiency impact of the changes in normalized cores.



Automatic Optimization

How did AI do better than the human optimization loop?

- **Quality**: Optimizing highly optimized code that was already optimized by humans.
- **Velocity**: 6.4K changes submitted in 1 year.
- **Coverage**: Optimized code that individually consumes too little resources to justify the human effort.

AI enables new systems approaches **today**.

Forward-Looking Directions

AI Reasoning Capabilities

Reasoning can unlock new systems capabilities. Consider deciding **which of two programs will complete first**:

```
p = new File("/tmp/book.txt");
for (i = 0; i < kNumPages; ++i) {
  convertToJPEG(p.ReadNext());
}
```

```
p = new File("/tmp/movie.avi");
for (i = 0; i < kNumFrames; ++i) {
  convertToJPEG(p.ReadNext());
}
```

To a compiler, these two programs look the same, but a human would be able to tell the likely answer.

LLM Response

Gemini

Gemini 2.0 Flash

Reasoning and Guess:

- The number of iterations is probably much higher for the movie program (`kNumFrames` >> `kNumPages`).
- The amount of data read per iteration (`p.readNext()`) is probably higher for the movie program (frame > page).
- The complexity and processing time of `convertToJPEG()` are probably significantly higher for movie frames.

Given all these factors, it's highly likely that the program processing the movie will take significantly longer. The sheer number of frames and the computational overhead of processing image data compared to text strongly suggest this.

Conclusion:

I would guess that the **program processing the movie (** `movie.avi` **)** will have the longer execution time.

What could systems do with these capabilities?

Catalyst #3:

GenAI enables new approaches that would not be possible without it.

Talk Outline

1 Conceptual Abstractions
Standardized ways for building learning into systems

2 ML Support in Systems
Best practices for deploying learning-based systems

3 Growing AI Capabilities
GenAI and other approaches

Talk Outline

1 **Conceptual Abstractions**
Standardized ways for building learning into systems

2 **ML Support in Systems**
Best practices for deploying learning-based systems

3 **Growing AI Capabilities**
GenAI and other approaches

Conclusion

Conclusion

Improved **abstractions** and **support** can create a positive feedback loop for ML for Systems adoption.

Many of the challenges are **similar to transitions the systems community has made before** (single-node to distributed systems, single-core to multi-core).

Growing **AI capabilities** generate new opportunities that would not have been possible before.

Has ML for Systems reached an inflection point?

All the ingredients are there.

What it will look like and when it will manifest depends on the people in this room.

Thank you.

Presented work by: Aditya Patil, Amir Yazdanbakhsh, Arif Merchant, Arman Hasanzadeh, Arun Chauhan, Berkin Ilbeyi, Chenxi Yang, Chris Kennelly, Chunlei Wang, Clifford Stein, Colin Raffel, David G. Andersen, Deniz Altinbüken, Diwakar Gupta, Don Schwarz, Florin Papa, Fred Lewis, Giulio Zhou, Hannah Lin, Jason Behmer, Jianheng Ling, Kathryn S. McKinley, Logan A. Bush, Maggie Nolan Edmonds, Maximilian Roquemore, Meeyoung Park, Michael Isard, Milad Hashemi, Mohammad Mahdi Javanmard, Mustafa Uysal, Parthasarathy Ranganathan, Phitchaya Mangpo Phothilimthana, Prakash Ramanan, Pratik Worah, Rajesh Kumar, Richard McDougall, Satish Chandra, Saurabh Kadekodi, Selene Moon, Soroush Ghodrati, Thomas Chestna, Tzu-Wei Yang, Ubaid Ullah Hafeez, Ulysse Beaugnon, Yajing Liu, Yan Li, Yawen Wang, Ye Zhao, Ying Liu, Yunchuan Kong, Yusuf Simonson + others.

Martin Maas
mmaas@google.com

Google DeepMind